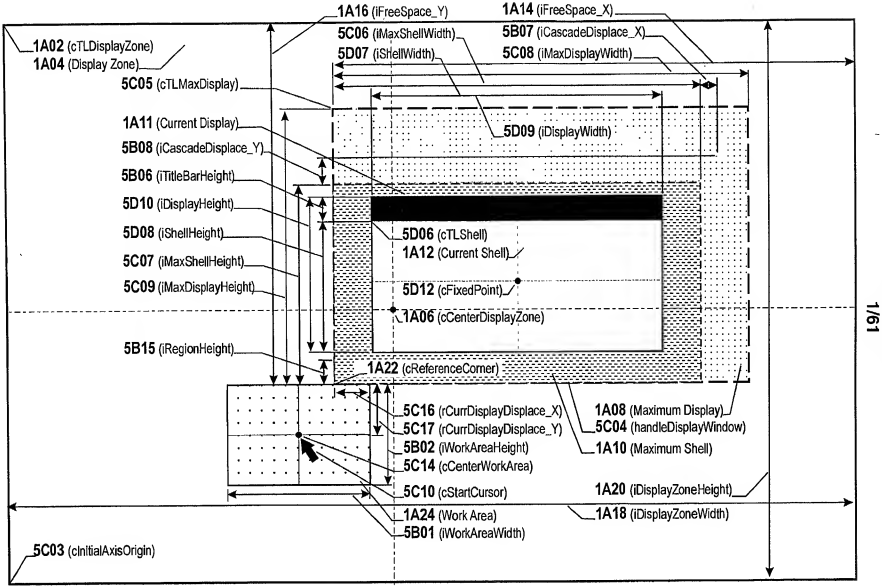


FIGURE 1A

Title: METHOD TO DISPLAY AND MANAGE COMPUTER POP-UP CONTROLS
Inventor: Stanley W. Driskell
Atty. Ref. No.: 8219-000006

NO CONSTRAINT:
iFreeSpace_X ≥ iMaxDisplayWidth
iFreeSpace_Y ≥ iMaxDisplayHeight

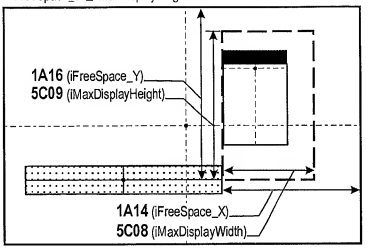


FIGURE 1B1

HORIZONTAL CONSTRAINT:
iFreeSpace_X < iMaxDisplayWidth
iFreeSpace_Y ≥ iMaxDisplayHeight

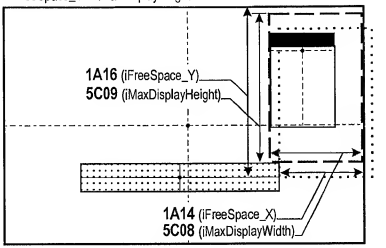


FIGURE 1B2

VERTICAL CONSTRAINT:
iFreeSpace_X ≥ iMaxDisplayWidth
iFreeSpace_Y < iMaxDisplayHeight

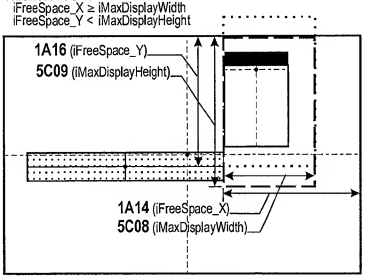


FIGURE 1B3

HORIZONTAL & VERTICAL CONSTRAINT:
iFreeSpace_X < iMaxDisplayWidth
iFreeSpace_Y < iMaxDisplayHeight

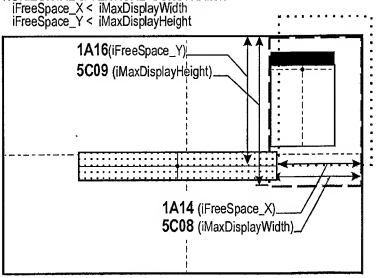


FIGURE 1B4

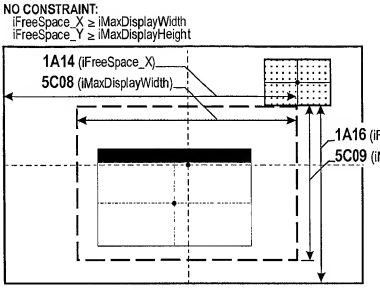


FIGURE 1C1

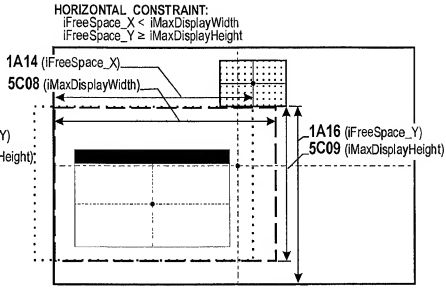


FIGURE 1C2

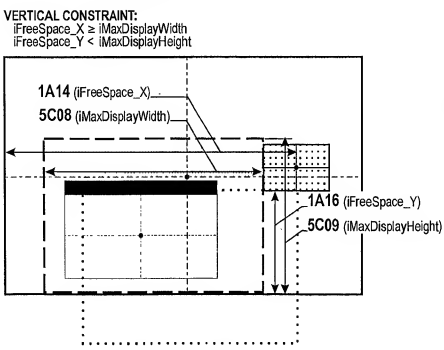


FIGURE 1C3

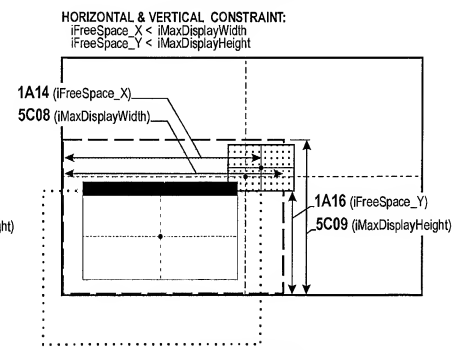


FIGURE 1C4

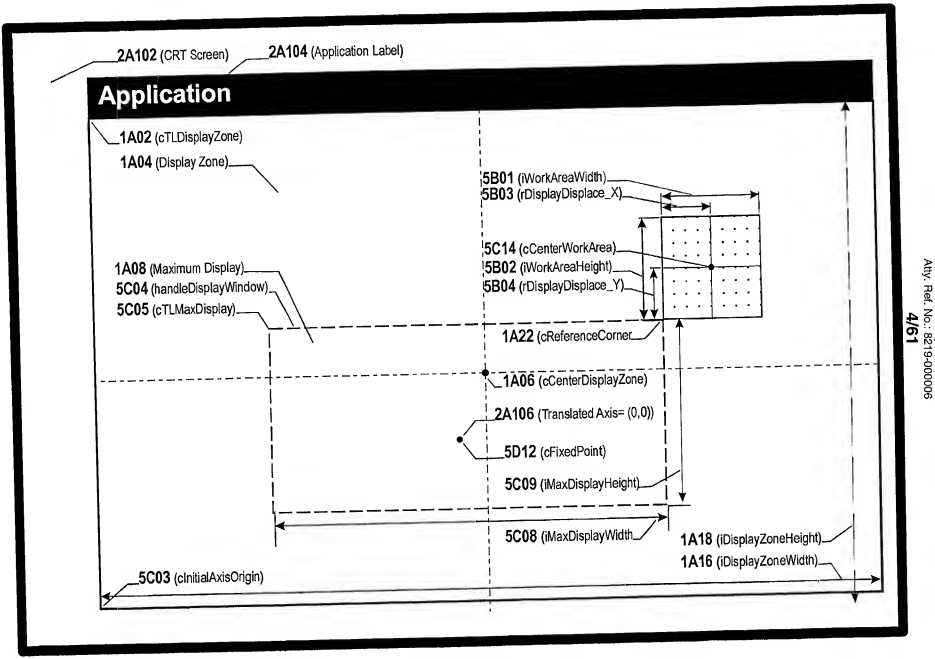


FIGURE 2A1

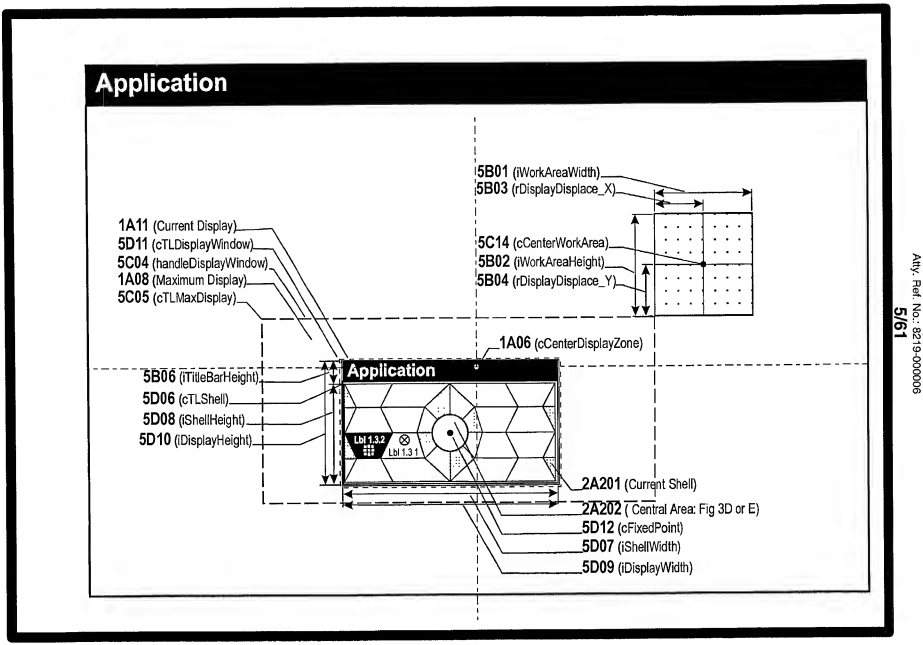


FIGURE 2A2

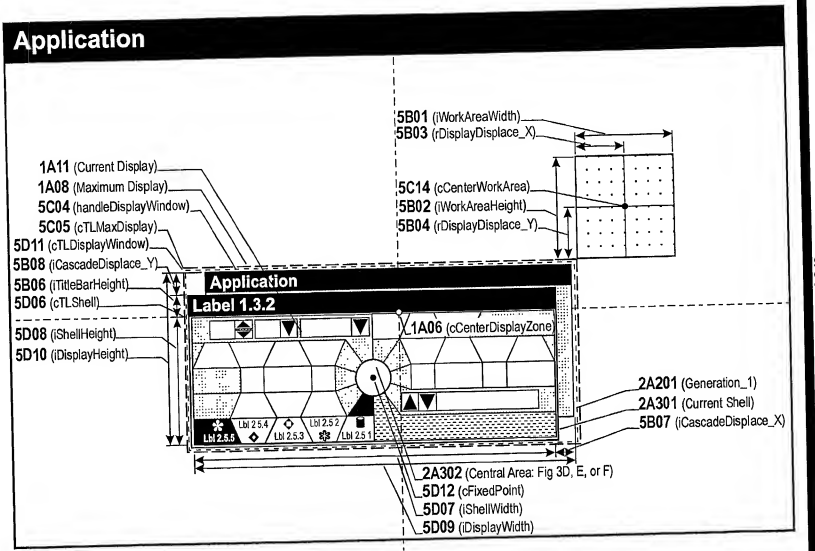

Application
5B01 (iWorkAreaWidth)
5B03 (rDisplayDisplace_X)
1A11 (Current Display)
1A08 (Maximum Display)
5C04 (handleDisplayWindow)
5C05 (cTLMaxDisplay)
5D11 (cTLDisplayWindow)
5B08 (iCascadeDisplace_Y)
5B06 (iTitleBarHeight)
5D06 (cTLShell)
5D08 (iShellHeight)
5D10 (iDisplayHeight)
5C14 (cCenterWorkArea)
5B02 (iWorkAreaHeight)
5B04 (rDisplayDisplace_Y)
Application
Label 1.3.2
1A06 (cCenterDisplayZone)
Lbl 2.5.4
Lbl 2.5.2
Lbl 2.5.5
Lbl 2.5.3
Lbl 2.5.1
2A201 (Generation_1)
2A301 (Current Shell)
5B07 (iCascadeDisplace_X)
2A302 (Central Area: Fig 3D, E, or F)
5D12 (cFixedPoint)
5D07 (iShellWidth)
5D09 (iDisplayWidth)

FIGURE 2A3

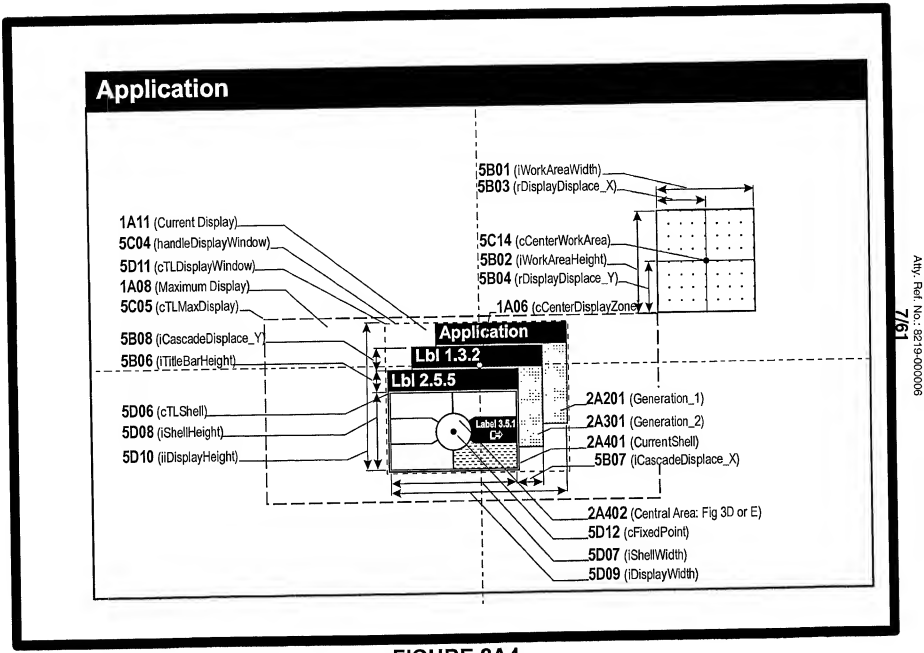


FIGURE 2A4

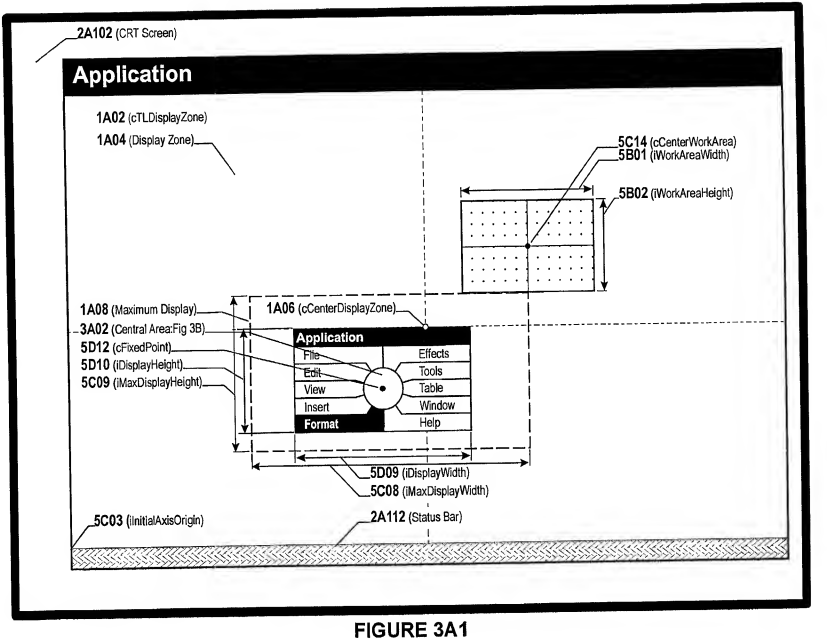


FIGURE 3A1

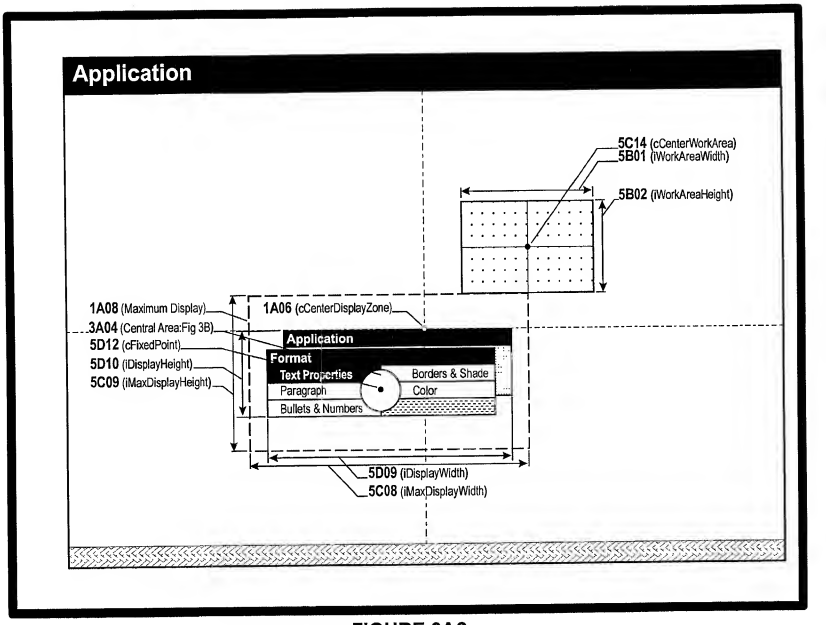


FIGURE 3A2

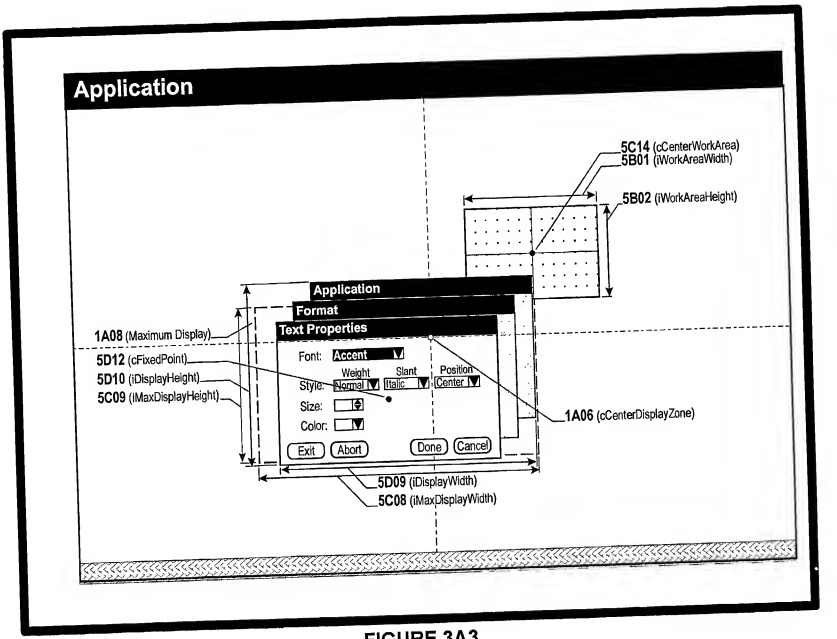


FIGURE 3A3

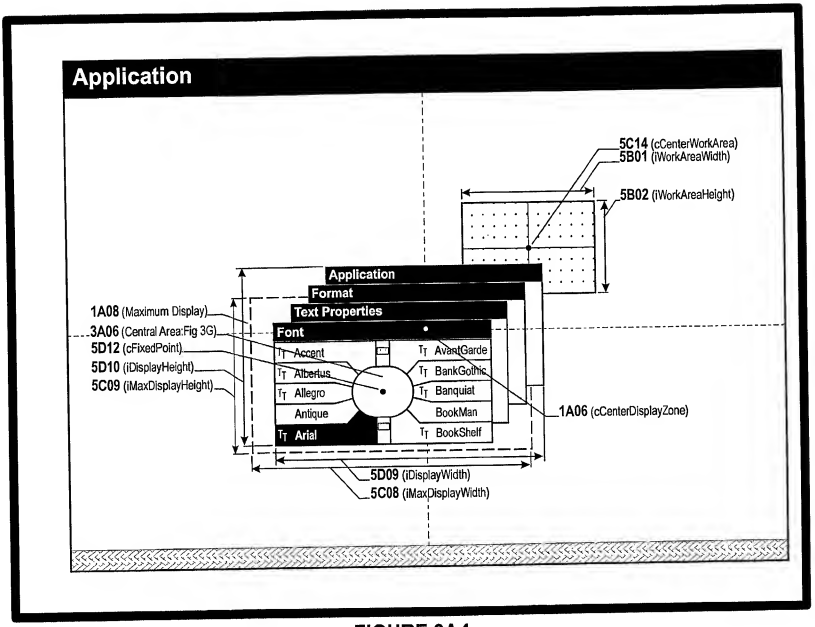


FIGURE 3A4

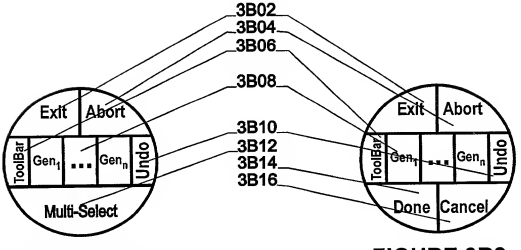


FIGURE 3B1

FIGURE 3B2

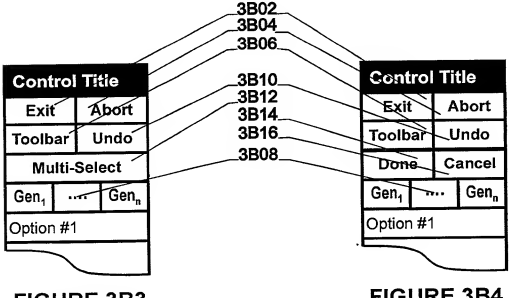


FIGURE 3B3

FIGURE 3B4

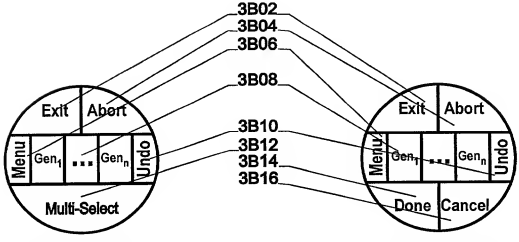


FIGURE 3C1

FIGURE 3C2

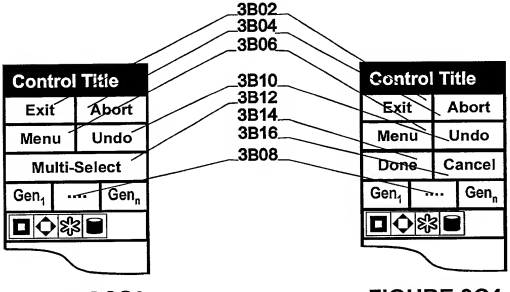


FIGURE 3C3

FIGURE 3C4

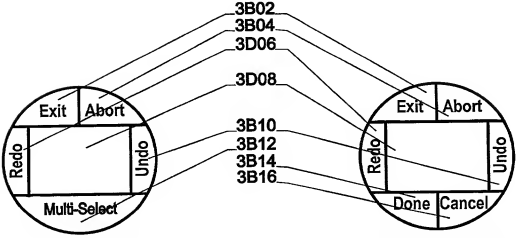


FIGURE 3D1

FIGURE 3D2

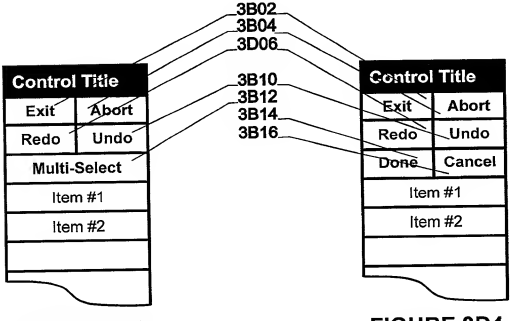


FIGURE 3D3

FIGURE 3D4

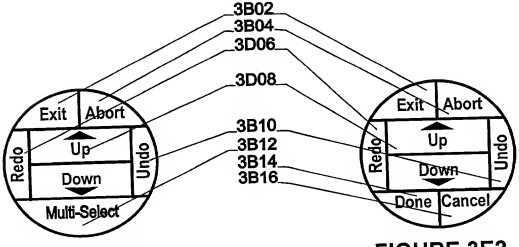


FIGURE 3E1

FIGURE 3E2

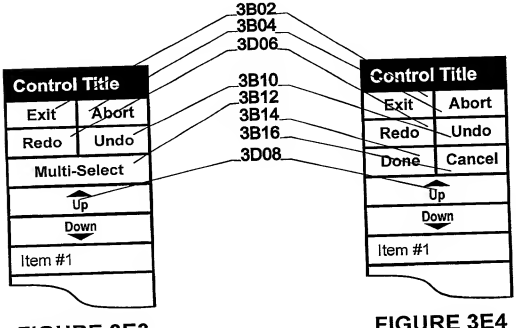


FIGURE 3E3

FIGURE 3E4

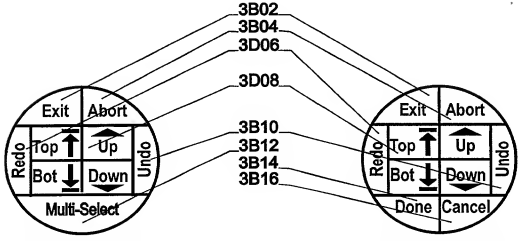


FIGURE 3F1

FIGURE 3F2

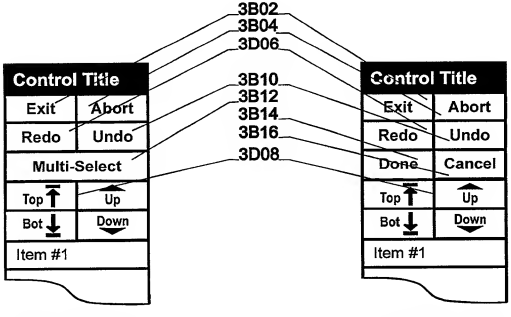


FIGURE 3F3

FIGURE 3F4

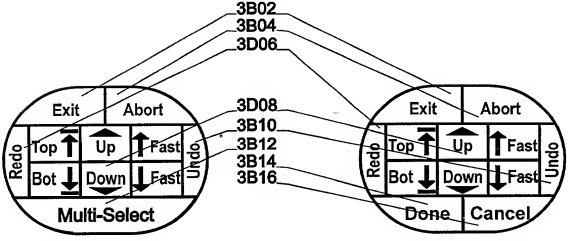


FIGURE 3G1

FIGURE 3G2

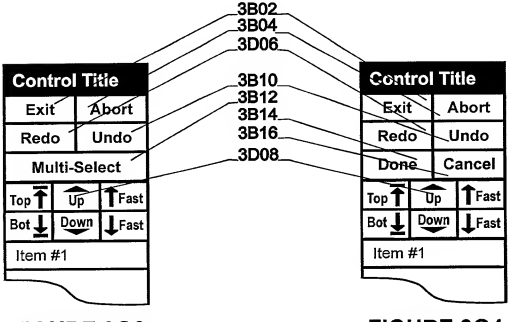


FIGURE 3G3

FIGURE 3G4

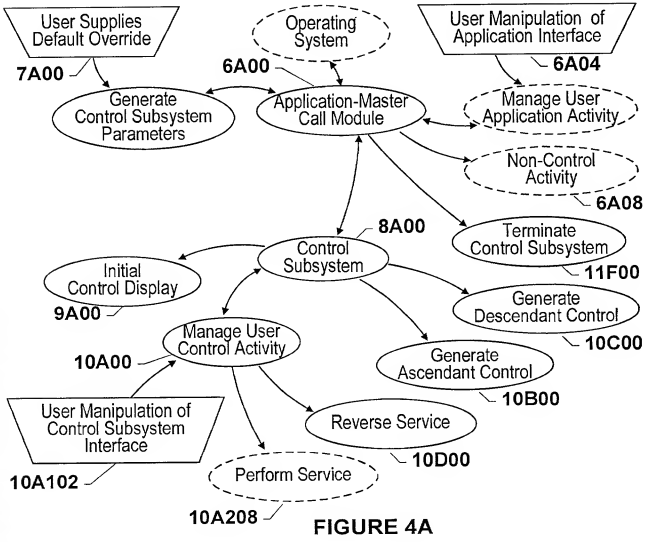


FIGURE 4A

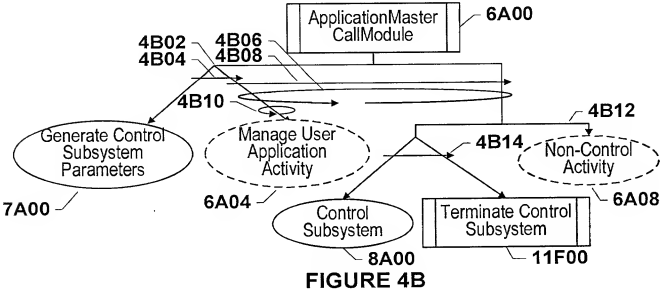


FIGURE 4B

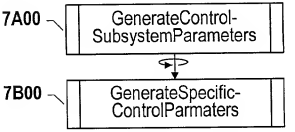


FIGURE 4C

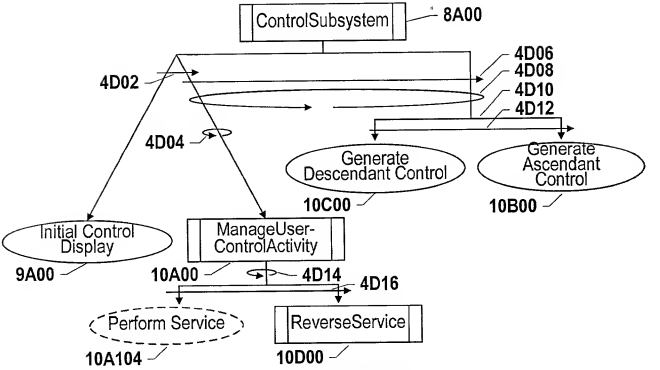


FIGURE 4D

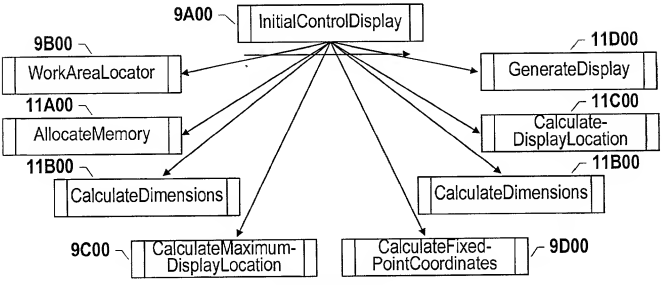


FIGURE 4E

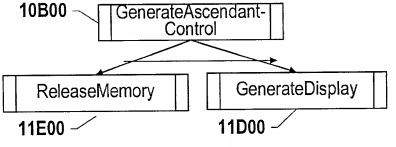


FIGURE 4F

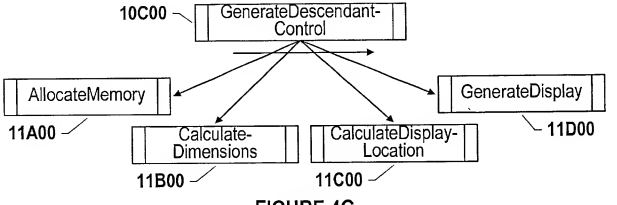
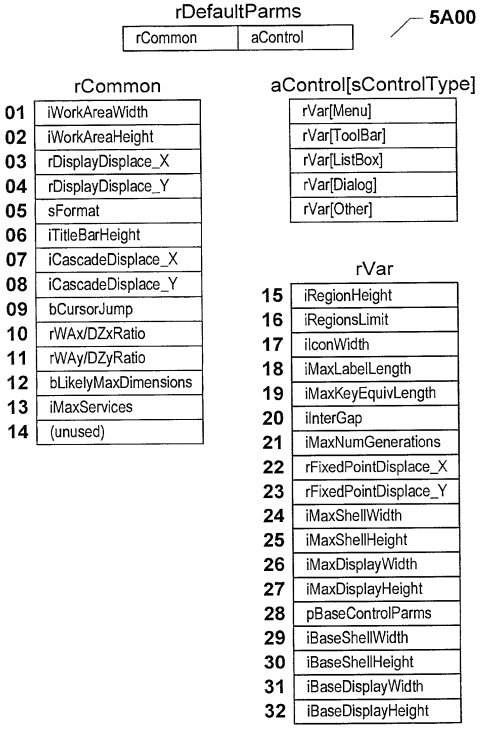


FIGURE 4G



DEFINITIONS: See definitions of Figure 5B

FIGURE 5A

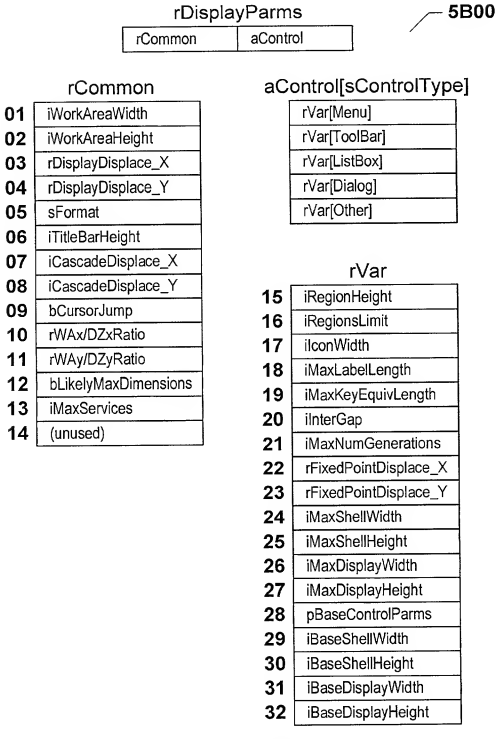


FIGURE 5B1

DEFINITIONS

rCommon: PARAMETER VALUES APPLICABLE TO ALL CONTROLS

01	iWorkAreaWidth	Width of implicit WorkArea.
02	iWorkAreaHeight	Height of implicit Work Area.
03	rDisplayDisplace_X	The per cent of one-half the work area width from which the reference corner of the maximim display is displaced from the Work Area center: -- Positive displacement is toward the Display Zone center. -- Negative displacement is away from the Display Zone center.
04	rDisplayDisplace_Y	The per cent of one-half the work area height from which the reference corner of the maximim display is displaced from the Work Area center. -- Positive displacement is toward the Display Zone center. -- Negative displacement is away from the Display Zone center.
05	sFormat	String identifying display format: "Traditional", "Spider", "Dialog" or "Other".
06	iTitleBarHeight	Height of title bar.
07	iCascadeDisplace_X	Horizontal displacement with which each successive ancestor of the control display is cascaded.
08	iCascadeDisplace_Y	Vertical displacement with which each successive of ancestorof the control display is cascaded.
09	bJumpCursor	Default disposal of cursor at termination of control display: -- TRUE -> DO move cursor to cStartCurtsor coordinates. -- FALSE-> DO NOT move cursor to cStartCurtsor coordinates.
10	rWAx/DZxRatio	Ratio of work-area/display-zone width that sets rDisplayDisplace_Y=0.0 and rDisplayDisplace_X=1.0;
11	rWAz/DZzRatio	Ratio of work-area/display-zone height that sets rDisplayDisplace_Y=1.0 and rDisplayDisplace_X=0.0;
12	bLikelyMaxDimensions	TRUE -> User supplies dimensions of likely largest control display, FALSE -> User supplies dimensions of the absolutely maximum display.
13	iMaxServices	Maximum number of services that will ever ever requested during a Contro Subsystem activation.

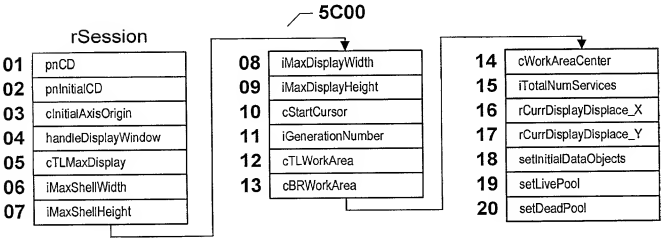
FIGURE 5B2

DEFINITIONS (Continued)

aControl[sFormat].rVar: PARAMETER VALUES SPECIFIC TO EACH CONTROL MANAGED

15 iRegionHeight	Height of area of display in which an item or a group of items is displayed.
16 iRegionsLimit	Max number of different items or groups of related items appearing on any single display.
17 iIconWidth	Width of any icon present.
18 iMaxLabelLength	Maximum length of any label appearing in a control.
19 iMaxKeyEquivLength	Maximum length of any key-equavilent symbols appearing in a control.
20 iInterGap	Spacing between identification elements in a region
21 iMaxNumGenerations	Maximum number of controls that can be present in a control path.
22 rFixedPointDisplace_X	Percent of iMaxShellWidth the cFixedFocusPoint.X is displaced rightward from top-left corner of the maximum shell.
23 rFixedPointDisplace_Y	Percent of iMaxShellHeight the cFixedFocusPoint.Y is displaced downward from top-left corner of the maximum shell.
24 iMaxShellWidth	Maximum width required for Menu & ToolBar shell. Null for other controls
25 iMaxShellHeight	Maximum Height required for Menu & ToolBar shell. Null for other controls.
26 iMaxDisplayWidth	Maximum width required for Menu & ToolBar display. Null for other controls
27 iMaxDisplayHeight	Maximum Height required for Menu & ToolBar display. Null for other controls.
28 pBaseControlParms	Pointer to the data structure defining the initial menu or toolbar displayed at control activation. This parameter is null for controls having parameters that can vary during different activations.
29 iBaseShellWidth	Width of initial Menu &r ToolBar shell. Null for other controls
30 iBaseShellHeight	Height of initial Menu & ToolBar shell. Null for other controls.
31 iBaseDisplayWidth	Width of initial Menu & ToolBar display. Null for other controls
32 iBaseDisplayHeight	Height of initial Menu & ToolBar display. Null for other controls.

FIGURE 5B3



DEFINITIONS

rCurrentSession: VARIABLES OF CURRENT ACTIVATION OF DISPLAY SUB-SYSTEM

01 pnCD Pointer to node of llnCD storing values fo the current display.

02 pInitialnAC Pointer to first node of the llnCD data structure.

03 cInitialAxisOrigin Screen coordinate pair of defining the axis origin at control subsystem activation.

04 handleDisplayWindow Pointer to data and procedures controlled by the operating system that manage the window containing the controldisplay.

05 cTLMaxDisplay Coordinate pair defining best location of top-left corner of the maximum display.

06 iMaxShellWidth Width of max shell that can be encountered given the initial display requested.

07 iMaxShellHeight Height of max shell that can be encountered given the initial display requested.

08 iMaxDisplayWidth Width of max display that can be encountered given the initial display requested.

09 iMaxDisplayHeight Height of max display that can be encountered given the initial display requested.

10 cStartCursor Coordinate pair defining the cursor location relative to cInitialAxisOrigin at control subsystem activation.

11 iGenerationNumber Number of generations of current toolbar display.

12 cTLWorkArea Coordinate pair defining the top-left corner of the current work area.

13 cBRWorkArea Coordinates of bottom-right corner of the current work area.

14 cWorkAreaCenter Coordinate pair defining the center of the current work area.

15 iTotalNumServices Number of services performed during the current control subsystem activation.

16 rCurrDisplayDisplace_X rDisplayDisplace_X parameter with allowance for extreme work area dimensions.

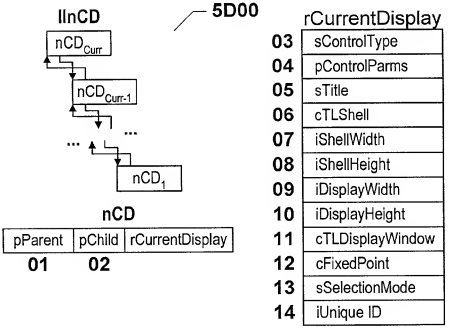
17 rCurrDisplayDisplace_Y rDisplayDisplace_Y parameter with allowance for extreme work area dimensions.

18 setInitialDataObjects Set of data objects existing control subsystem activated.

19 setLivePool Accumulating set of data objects created by user manipulation of controls during current activation of the control subsystem.

20 setDeadPool Accumulating set of data objects destroyed by user manipulation of controls during current activation of the control subsystem.

FIGURE 5C



DEFINITIONS

01 pParent — Pointer to parent of current nCD node
02 pChild — Pointer to child of current nCD node.

rCurrentDisplay: VARIABLE VALUES TO MANAGE CURRENT DISPLAY

03 sControlType — Type of control managed by current llnCD node:
"Menu", "ToolBar", "ListBox", "Dialog", "Other".

04 pControlParms — Pointer to system data structure defining parameters of the control identified by the current nCD node.

05 sTitle — String containing label appearing inTitleBar

06 cTLShell — Coordinate pair defining the top-left corner of the shell of the current control.

07 iShellWidth — Width of the current shell.

08 iShellHeight — Height of the current shell.

09 iDisplayWidth — Width of the current display.

10 iDisplayHeight — Height of the current display.

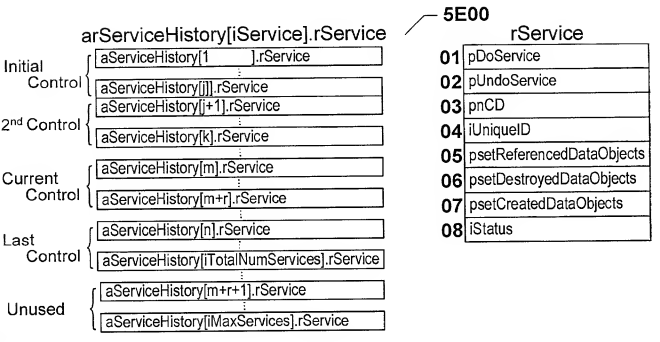
11 cTLDisplayWindow — Coordinate pair defining the top-left corner of the current display.

12 cFixedPoint — Coordinate pair defining the current fixed point.

13 sSelectionMode — Designates kind of selection from control is permitted
- -- Single -> single selection is imposed by design.
- -- User ->"MultiSelect" displayed.
  - * If "Multi-Sect" not selected, user is constrained to single selection.
  - * If "Multi-Sect" selected, the multi-select area is reconfigured to "Done|Cancel", and the user is not constrained to single selection..
- -- Multiple -> multiple selection is permitted.

14 iUniqueID — The unique identification number of specific control defined by current llnCD node.

FIGURE 5D



DEFINITIONS

rService: VARIABLE VALUES TO MANAGE ARBITRARY UNDO

01 pDoService Pointer to processes that perform the Xth service of the current Control Subsystem activation.

02 pUndoService Pointer to processes that perform the reverse capability of the Xth service of the current Control Subsystem activation.

03 pnCD Pointer to InCD node defining the control from which the Xth service of the current control subsystem activation was requested.

04 iUniqueID Unique identification number of the control identified by the current node.

05 psetReferencedDataObjects
Pointer to a set identifying all data objects utilized during performance of the Xth service excluding reference to any data object(s) created by that service.

06 psetDestroyedDataObjects
Pointer to a set identifying all data objects destroyed during performance of the Xth service.

07 psetCreatedDataObjects
Pointer to a set identifying all data objects created during performance of the Xth service.

08 iStatus Indicator to the current state of the Xth service:
+1 -> service has not been reversed
0 -> service is temporarily reversed
-1 -> service currently reversed because of dependence on a one or more data objects created by prior service(s) that have been reversed.
-2 -> service is permanently reversed

FIGURE 5E

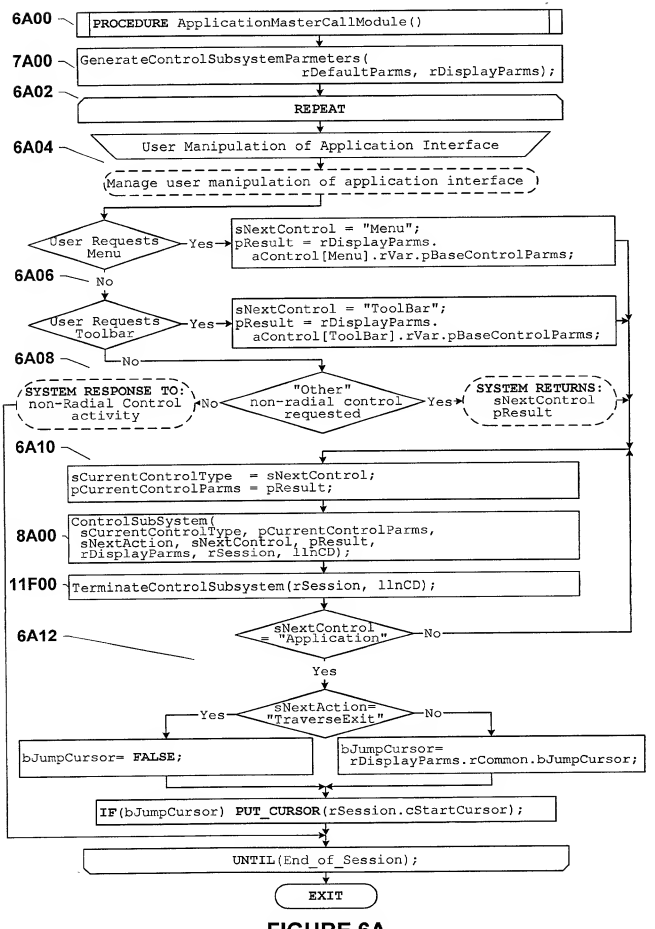

6A00
PROCEDURE ApplicationMasterCallModule()
7A00
GenerateControlSubsystemParmeters(rDefaultParms, rDisplayParms);
6A02
REPEAT
6A04
User Manipulation of Application Interface
Manage user manipulation of application interface
User Requests Menu
Yes
sNextControl = "Menu";
pResult = rDisplayParms.
aControl[Menu].rVar.pBaseControlParms;
6A06
No
User Requests Toolbar
Yes
sNextControl = "ToolBar";
pResult = rDisplayParms.
aControl[ToolBar].rVar.pBaseControlParms;
6A08
No
SYSTEM RESPONSE TO: non-Radial Control activity
No
"Other" non-radial control requested
Yes
SYSTEM RETURNS: sNextControl pResult
6A10
sCurrentControlType = sNextControl;
pCurrentControlParms = pResult;
8A00
ControlSubSystem(sCurrentControlType, pCurrentControlParms, sNextAction, sNextControl, pResult, rDisplayParms, rSession, llnCD);
11F00
TerminateControlSubsystem(rSession, llnCD);
6A12
sNextControl = "Application"
No
Yes
sNextAction= "TraverseExit"
Yes
No
bJumpCursor= FALSE;
bJumpCursor= rDisplayParms.rCommon.bJumpCursor;
IF(bJumpCursor) PUT_CURSOR(rSession.cStartCursor);
UNTIL(End_of_Session);
EXIT

FIGURE 6A

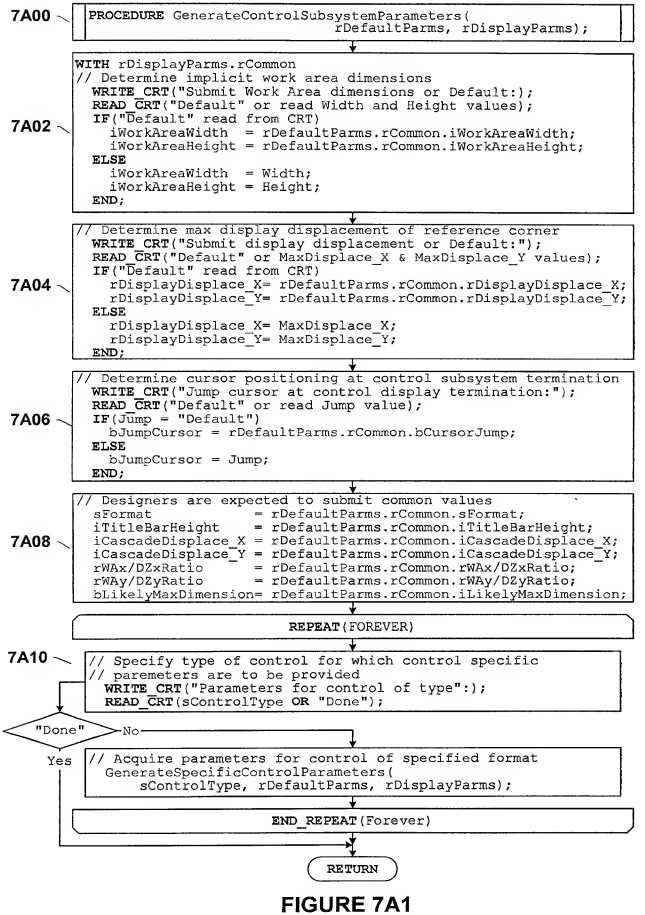

7A00
PROCEDURE GenerateControlSubsystemParameters(rDefaultParms, rDisplayParms);
WITH rDisplayParms.rCommon
// Determine implicit work area dimensions
WRITE_CRT("Submit Work Area dimensions or Default:);
READ_CRT("Default" or read Width and Height values);
7A02
IF("Default" read from CRT)
iWorkAreaWidth = rDefaultParms.rCommon.iWorkAreaWidth;
iWorkAreaHeight = rDefaultParms.rCommon.iWorkAreaHeight;
ELSE
iWorkAreaWidth = Width;
iWorkAreaHeight = Height;
END;
// Determine max display displacement of reference corner
WRITE_CRT("Submit display displacement or Default:");
READ_CRT("Default" or MaxDisplace_X & MaxDisplace_Y values);
7A04
IF("Default" read from CRT)
rDisplayDisplace_X= rDefaultParms.rCommon.rDisplayDisplace_X;
rDisplayDisplace_Y= rDefaultParms.rCommon.rDisplayDisplace_Y;
ELSE
rDisplayDisplace_X= MaxDisplace_X;
rDisplayDisplace_Y= MaxDisplace_Y;
END;
// Determine cursor positioning at control subsystem termination
WRITE_CRT("Jump cursor at control display termination:");
READ_CRT("Default" or read Jump value);
7A06
IF(Jump = "Default")
bJumpCursor = rDefaultParms.rCommon.bCursorJump;
ELSE
bJumpCursor = Jump;
END;
// Designers are expected to submit common values
sFormat = rDefaultParms.rCommon.sFormat;
iTitleBarHeight = rDefaultParms.rCommon.iTitleBarHeight;
7A08
iCascadeDisplace_X = rDefaultParms.rCommon.iCascadeDisplace_X;
iCascadeDisplace_Y = rDefaultParms.rCommon.iCascadeDisplace_Y;
rWAx/DZxRatio = rDefaultParms.rCommon.rWAx/DZxRatio;
rWAy/DZyRatio = rDefaultParms.rCommon.rWAy/DZyRatio;
bLikelyMaxDimension= rDefaultParms.rCommon.iLikelyMaxDimension;
REPEAT(FOREVER)
7A10
// Specify type of control for which control specific
// paremeters are to be provided
WRITE_CRT("Parameters for control of type":);
READ_CRT(sControlType OR "Done");
"Done"
No
Yes
// Acquire parameters for control of specified format
GenerateSpecificControlParameters(
sControlType, rDefaultParms, rDisplayParms);
END_REPEAT(Forever)
RETURN

FIGURE 7A1

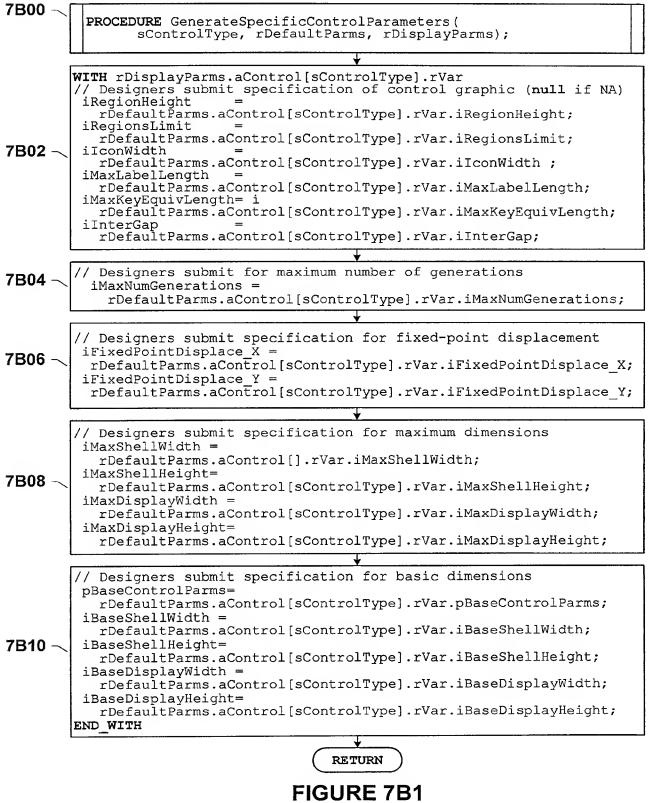

7B00
PROCEDURE GenerateSpecificControlParameters(
sControlType, rDefaultParms, rDisplayParms);
7B02
WITH rDisplayParms.aControl[sControlType].rVar
// Designers submit specification of control graphic (null if NA)
iRegionHeight =
rDefaultParms.aControl[sControlType].rVar.iRegionHeight;
iRegionsLimit =
rDefaultParms.aControl[sControlType].rVar.iRegionsLimit;
iIconWidth =
rDefaultParms.aControl[sControlType].rVar.iIconWidth ;
iMaxLabelLength =
rDefaultParms.aControl[sControlType].rVar.iMaxLabelLength;
iMaxKeyEquivLength= i
rDefaultParms.aControl[sControlType].rVar.iMaxKeyEquivLength;
iInterGap =
rDefaultParms.aControl[sControlType].rVar.iInterGap;
7B04
// Designers submit for maximum number of generations
iMaxNumGenerations =
rDefaultParms.aControl[sControlType].rVar.iMaxNumGenerations;
7B06
// Designers submit specification for fixed-point displacement
iFixedPointDisplace_X =
rDefaultParms.aControl[sControlType].rVar.iFixedPointDisplace_X;
iFixedPointDisplace_Y =
rDefaultParms.aControl[sControlType].rVar.iFixedPointDisplace_Y;
7B08
// Designers submit specification for maximum dimensions
iMaxShellWidth =
rDefaultParms.aControl[].rVar.iMaxShellWidth;
iMaxShellHeight=
rDefaultParms.aControl[sControlType].rVar.iMaxShellHeight;
iMaxDisplayWidth =
rDefaultParms.aControl[sControlType].rVar.iMaxDisplayWidth;
iMaxDisplayHeight=
rDefaultParms.aControl[sControlType].rVar.iMaxDisplayHeight;
7B10
// Designers submit specification for basic dimensions
pBaseControlParms=
rDefaultParms.aControl[sControlType].rVar.pBaseControlParms;
iBaseShellWidth =
rDefaultParms.aControl[sControlType].rVar.iBaseShellWidth;
iBaseShellHeight=
rDefaultParms.aControl[sControlType].rVar.iBaseShellHeight;
iBaseDisplayWidth =
rDefaultParms.aControl[sControlType].rVar.iBaseDisplayWidth;
iBaseDisplayHeight=
rDefaultParms.aControl[sControlType].rVar.iBaseDisplayHeight;
END_WITH
RETURN

FIGURE 7B1

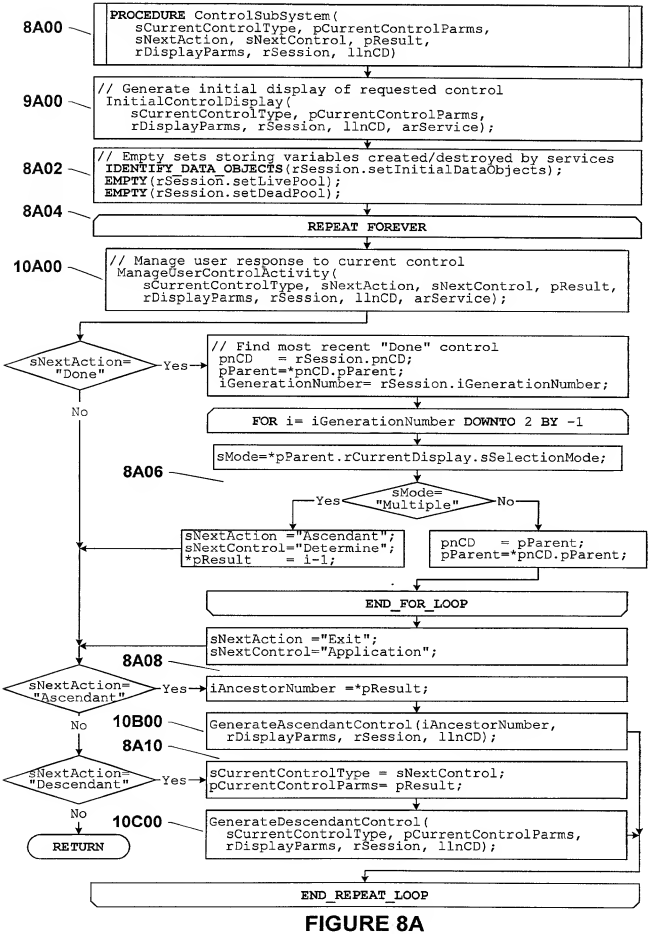

8A00
PROCEDURE ControlSubSystem(
sCurrentControlType, pCurrentControlParms,
sNextAction, sNextControl, pResult,
rDisplayParms, rSession, llnCD)
9A00
// Generate initial display of requested control
InitialControlDisplay(
sCurrentControlType, pCurrentControlParms,
rDisplayParms, rSession, llnCD, arService);
8A02
// Empty sets storing variables created/destroyed by services
IDENTIFY_DATA_OBJECTS(rSession.setInitialDataObjects);
EMPTY(rSession.setLivePool);
EMPTY(rSession.setDeadPool);
8A04
REPEAT FOREVER
10A00
// Manage user response to current control
ManageUserControlActivity(
sCurrentControlType, sNextAction, sNextControl, pResult,
rDisplayParms, rSession, llnCD, arService);
sNextAction= "Done"
Yes
// Find most recent "Done" control
pnCD = rSession.pnCD;
pParent=*pnCD.pParent;
iGenerationNumber= rSession.iGenerationNumber;
No
FOR i= iGenerationNumber DOWNTO 2 BY -1
sMode=*pParent.rCurrentDisplay.sSelectionMode;
8A06
sMode= "Multiple"
Yes
No
sNextAction ="Ascendant";
sNextControl="Determine";
*pResult = i-1;
pnCD = pParent;
pParent=*pnCD.pParent;
END_FOR_LOOP
sNextAction ="Exit";
sNextControl="Application";
8A08
sNextAction= "Ascendant"
Yes
iAncestorNumber =*pResult;
No
10B00
GenerateAscendantControl(iAncestorNumber,
rDisplayParms, rSession, llnCD);
8A10
sNextAction= "Descendant"
Yes
sCurrentControlType = sNextControl;
pCurrentControlParms= pResult;
No
10C00
GenerateDescendantControl(
sCurrentControlType, pCurrentControlParms,
rDisplayParms, rSession, llnCD);
RETURN
END_REPEAT_LOOP

FIGURE 8A

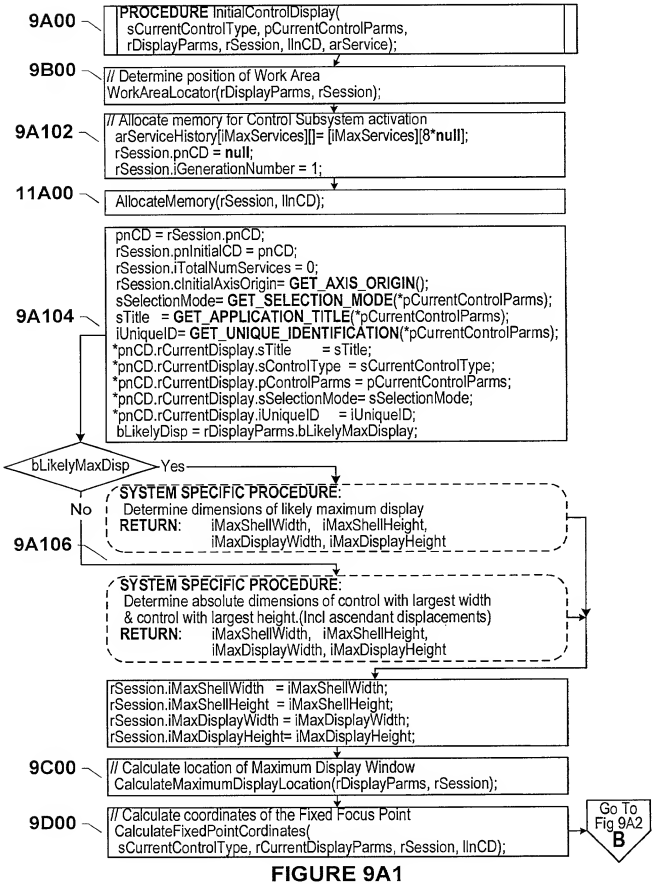

9A00
PROCEDURE InitialControlDisplay(
sCurrentControlType, pCurrentControlParms,
rDisplayParms, rSession, llnCD, arService);
9B00
// Determine position of Work Area
WorkAreaLocator(rDisplayParms, rSession);
9A102
// Allocate memory for Control Subsystem activation
arServiceHistory[iMaxServices][]= [iMaxServices][8*null];
rSession.pnCD = null;
rSession.iGenerationNumber = 1;
11A00
AllocateMemory(rSession, llnCD);
9A104
pnCD = rSession.pnCD;
rSession.pnInitialCD = pnCD;
rSession.iTotalNumServices = 0;
rSession.clnitialAxisOrigin= GET_AXIS_ORIGIN();
sSelectionMode= GET_SELECTION_MODE(*pCurrentControlParms);
sTitle = GET_APPLICATION_TITLE(*pCurrentControlParms);
iUniqueID= GET_UNIQUE_IDENTIFICATION(*pCurrentControlParms);
*pnCD.rCurrentDisplay.sTitle = sTitle;
*pnCD.rCurrentDisplay.sControlType = sCurrentControlType;
*pnCD.rCurrentDisplay.pControlParms = pCurrentControlParms;
*pnCD.rCurrentDisplay.sSelectionMode= sSelectionMode;
*pnCD.rCurrentDisplay.iUniqueID = iUniqueID;
bLikelyDisp = rDisplayParms.bLikelyMaxDisplay;
bLikelyMaxDisp
Yes
No
SYSTEM SPECIFIC PROCEDURE:
Determine dimensions of likely maximum display
RETURN: iMaxShellWidth, iMaxShellHeight,
iMaxDisplayWidth, iMaxDisplayHeight
9A106
SYSTEM SPECIFIC PROCEDURE:
Determine absolute dimensions of control with largest width
& control with largest height.(Incl ascendant displacements)
RETURN: iMaxShellWidth, iMaxShellHeight,
iMaxDisplayWidth, iMaxDisplayHeight
rSession.iMaxShellWidth = iMaxShellWidth;
rSession.iMaxShellHeight = iMaxShellHeight;
rSession.iMaxDisplayWidth = iMaxDisplayWidth;
rSession.iMaxDisplayHeight= iMaxDisplayHeight;
9C00
// Calculate location of Maximum Display Window
CalculateMaximumDisplayLocation(rDisplayParms, rSession);
9D00
// Calculate coordinates of the Fixed Focus Point
CalculateFixedPointCordinates(
sCurrentControlType, rCurrentDisplayParms, rSession, llnCD);
Go To
Fig 9A2
B

FIGURE 9A1

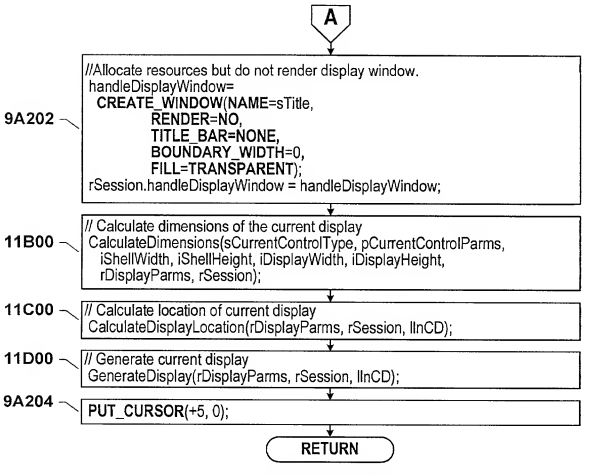

A
//Allocate resources but do not render display window.
handleDisplayWindow=
CREATE_WINDOW(NAME=sTitle,
RENDER=NO,
TITLE_BAR=NONE,
BOUNDARY_WIDTH=0,
FILL=TRANSPARENT);
rSession.handleDisplayWindow = handleDisplayWindow;
9A202
// Calculate dimensions of the current display
CalculateDimensions(sCurrentControlType, pCurrentControlParms,
iShellWidth, iShellHeight, iDisplayWidth, iDisplayHeight,
rDisplayParms, rSession);
11B00
// Calculate location of current display
CalculateDisplayLocation(rDisplayParms, rSession, llnCD);
11C00
// Generate current display
GenerateDisplay(rDisplayParms, rSession, llnCD);
11D00
PUT_CURSOR(+5, 0);
9A204
RETURN

FIGURE 9A2

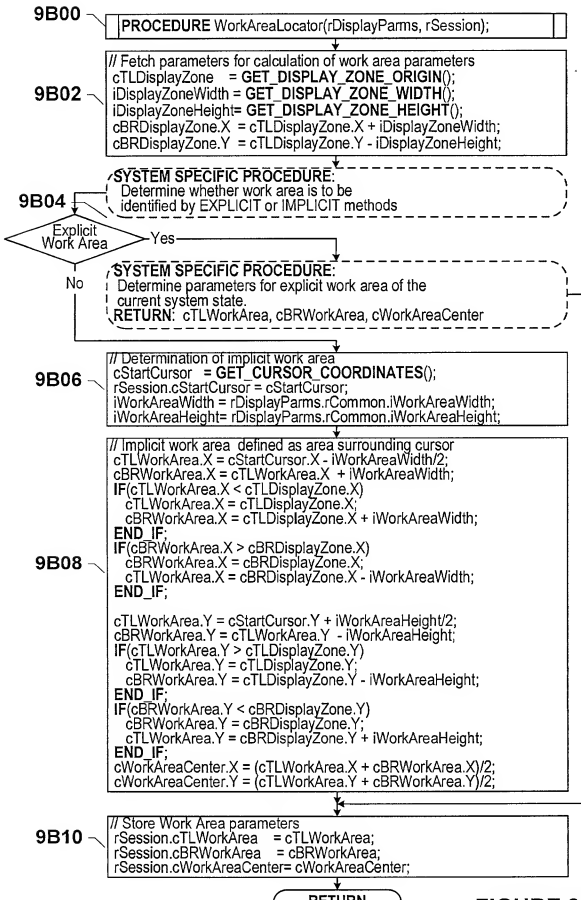


FIGURE 9B

```
PROGRAM CalculateMaximumDisplayLocation( rDisplayParms, rSession);
```
9C00

```
cTLDisplayZone    = GET_DISPLAY_ZONE_ORIGIN();
iDisplayZoneWidth = GET_DISPLAY_ZONE_WIDTH();
iDisplayZoneHeight= GET_DISPLAY_ZONE_HEIGHT();
cBRDisplayZone.X= cTLDisplayZone.X + iDisplayZoneWidth;
cBRDisplayZone.Y= cTLDisplayZone.Y - iDisplayZoneHeight;
```
9C102

```
//Determine paramaters to position Max Display reference corner
iMaxDisplayWidth  = rSession.cMaxDisplayWidth;
iMaxDisplayHeight = rSession.cMaxDisplayHeight;
cTLWorkArea       = rSession.cTLWorkArea;
cBRWorkArea       = rSession.cBRWorkArea;
iWorkAreaWidth    = cBRWorkArea.X - cTLWorkArea.X;
iWorkAreaHeight   = cTLWorkArea.Y - cBRWorkArea.Y;
rWAx/DZxRatio     = rDisplayParms.rCommon.rWAx/DZxRatio;
rWAy/DZyRatio     = rDisplayParms.rCommon.rWAy/DZyRatio;
IF(iWorkAreaWidth/iDisplayZoneWidth > rWAx/DZxRatio)
  rDisplayDisplace_X= 0.0;    rDisplayDisplace_Y= 1.0;
ELSEIF(iWorkAreaHeight/iDisplayZoneHeight > rWAy/DZyRatio)
  rDisplayDisplace_X= 1.0;    rDisplayDisplace_Y= 0.0;
ELSE
  rDisplayDisplace_X= rDisplayParms.rCommon.rDisplayDisplace_X;
  rDisplayDisplace_Y= rDisplayParms.rCommon.rDisplayDisplace_Y;
END_IF;
rSession.rCurrDisplayDisplace_X= rDisplayDisplace_X;
rSession.rCurrDisplayDisplace_Y= rDisplayDisplace_Y;
```
9C104

```
// Determine direction of Center Seeking:
IF((cWorkAreaCenter.X >= cDisplayZoneCenter.X)  AND
  (cWorkAreaCenter.Y >= cDisplayZoneCenter.Y))
    sCenterSeeking= "DownLeft";
IF((cWorkAreaCenter.X >= cDisplayZoneCenter.X)  AND
  (cWorkAreaCenter.Y <  cDisplayZoneCenter.Y))
    sCenterSeeking= "UpLeft";
IF((cWorkAreaCenter.X <  cDisplayZoneCenter.X)  AND
  (cWorkAreaCenter.Y >= cDisplayZoneCenter.Y))
    sCenterSeeking= "DownRight";
IF((cWorkAreaCenter.X <  cDisplayZoneCenter.X)  AND
  (cWorkAreaCenter.Y <  cDisplayZoneCenter.Y));
    sCenterSeeking= "UpRight";
```
9C106

```
// Calc optimum location of max control display reference corner
iDisplace_X= rDisplayDisplace_X*WorkAreaWidth/2;
iDisplace_Y= rDisplayDisplace_Y*WorkAreaHeight/2;
CASE(sCenterSeeking)
 "DownLeft":   cOpt.X= cWorkAreaCenter.X-iDisplace_X;
               cOpt.Y= cWorkAreaCenter.Y-iDisplace_Y;
 "UpLeft":     cOpt.X= cWorkAreaCenter.X-iDisplace_X;
               cOpt.Y= cWorkAreaCenter.Y+iDisplace_Y;
 "DownRight":  cOpt.X= cWorkAreaCenter.X+iDisplace_X;
               cOpt.Y= cWorkAreaCenter.Y-iDisplace_Y;
 "UpRight":    cOpt.X= cWorkAreaCenter.X+iDisplace_X;
               cOpt.Y= cWorkAreaCenter.Y+iDisplace_Y;
END_CASE;
```
9C108

Go To Fig 9C2 A

FIGURE 9C1

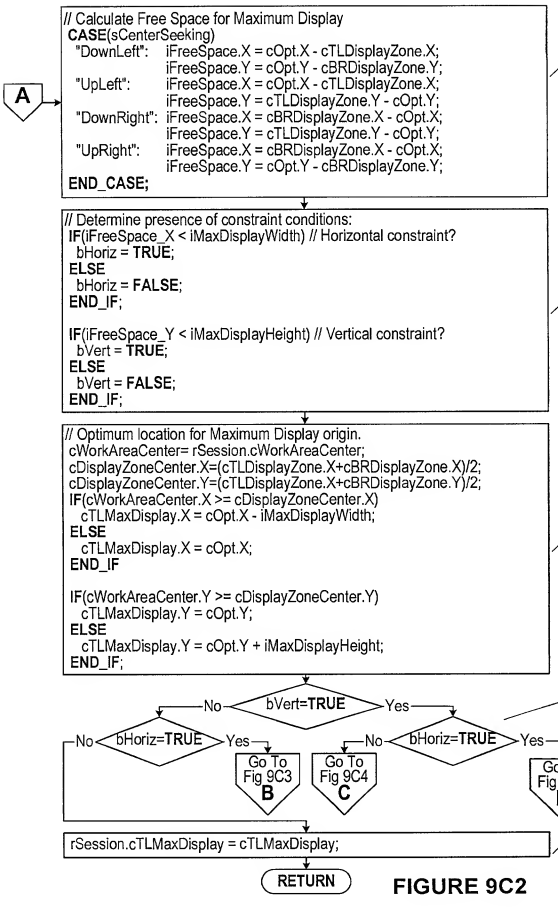

A
// Calculate Free Space for Maximum Display
CASE(sCenterSeeking)
"DownLeft": iFreeSpace.X = cOpt.X - cTLDisplayZone.X;
iFreeSpace.Y = cOpt.Y - cBRDisplayZone.Y;
"UpLeft": iFreeSpace.X = cOpt.X - cTLDisplayZone.X;
iFreeSpace.Y = cTLDisplayZone.Y - cOpt.Y;
"DownRight": iFreeSpace.X = cBRDisplayZone.X - cOpt.X;
iFreeSpace.Y = cTLDisplayZone.Y - cOpt.Y;
"UpRight": iFreeSpace.X = cBRDisplayZone.X - cOpt.X;
iFreeSpace.Y = cOpt.Y - cBRDisplayZone.Y;
END_CASE;
9C202
// Determine presence of constraint conditions:
IF(iFreeSpace_X < iMaxDisplayWidth) // Horizontal constraint?
bHoriz = TRUE;
ELSE
bHoriz = FALSE;
END_IF;
IF(iFreeSpace_Y < iMaxDisplayHeight) // Vertical constraint?
bVert = TRUE;
ELSE
bVert = FALSE;
END_IF;
9C204
// Optimum location for Maximum Display origin.
cWorkAreaCenter= rSession.cWorkAreaCenter;
cDisplayZoneCenter.X=(cTLDisplayZone.X+cBRDisplayZone.X)/2;
cDisplayZoneCenter.Y=(cTLDisplayZone.X+cBRDisplayZone.Y)/2;
IF(cWorkAreaCenter.X >= cDisplayZoneCenter.X)
cTLMaxDisplay.X = cOpt.X - iMaxDisplayWidth;
ELSE
cTLMaxDisplay.X = cOpt.X;
END_IF
IF(cWorkAreaCenter.Y >= cDisplayZoneCenter.Y)
cTLMaxDisplay.Y = cOpt.Y;
ELSE
cTLMaxDisplay.Y = cOpt.Y + iMaxDisplayHeight;
END_IF;
9C206
bVert=TRUE
No
Yes
bHoriz=TRUE
No
Yes
bHoriz=TRUE
No
Yes
9C208
Go To Fig 9C3 B
Go To Fig 9C4 C
Go To Fig 9C5 D
rSession.cTLMaxDisplay = cTLMaxDisplay;
9C210
RETURN

FIGURE 9C2

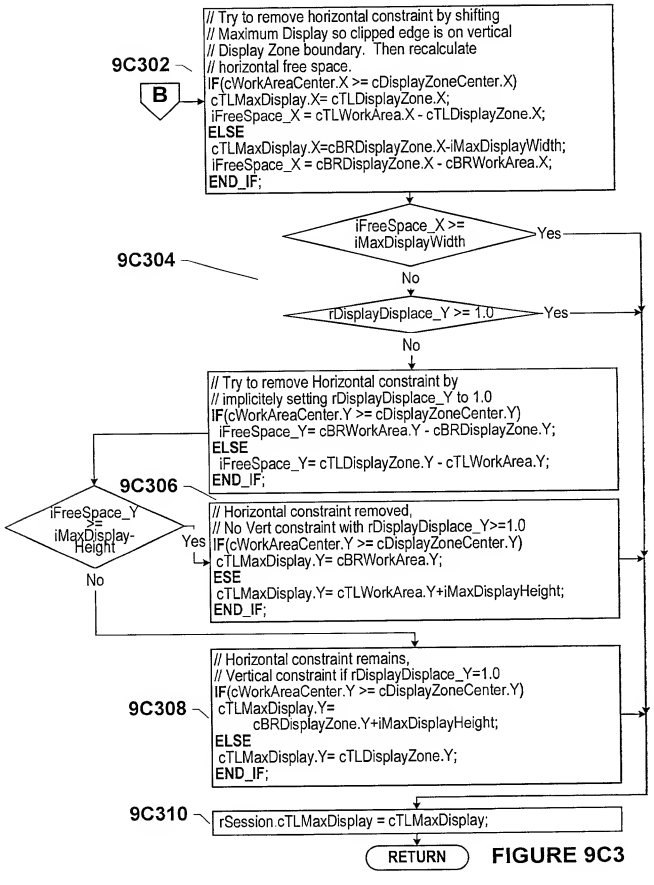

9C302
B
// Try to remove horizontal constraint by shifting
// Maximum Display so clipped edge is on vertical
// Display Zone boundary. Then recalculate
// horizontal free space.
IF(cWorkAreaCenter.X >= cDisplayZoneCenter.X)
cTLMaxDisplay.X= cTLDisplayZone.X;
iFreeSpace_X = cTLWorkArea.X - cTLDisplayZone.X;
ELSE
cTLMaxDisplay.X=cBRDisplayZone.X-iMaxDisplayWidth;
iFreeSpace_X = cBRDisplayZone.X - cBRWorkArea.X;
END_IF;
iFreeSpace_X >= iMaxDisplayWidth
Yes
9C304
No
rDisplayDisplace_Y >= 1.0
Yes
No
// Try to remove Horizontal constraint by
// implicitely setting rDisplayDisplace_Y to 1.0
IF(cWorkAreaCenter.Y >= cDisplayZoneCenter.Y)
iFreeSpace_Y= cBRWorkArea.Y - cBRDisplayZone.Y;
ELSE
iFreeSpace_Y= cTLDisplayZone.Y - cTLWorkArea.Y;
END_IF;
9C306
iFreeSpace_Y >= iMaxDisplay-Height
Yes
No
// Horizontal constraint removed,
// No Vert constraint with rDisplayDisplace_Y>=1.0
IF(cWorkAreaCenter.Y >= cDisplayZoneCenter.Y)
cTLMaxDisplay.Y= cBRWorkArea.Y;
ESE
cTLMaxDisplay.Y= cTLWorkArea.Y+iMaxDisplayHeight;
END_IF;
// Horizontal constraint remains,
// Vertical constraint if rDisplayDisplace_Y=1.0
IF(cWorkAreaCenter.Y >= cDisplayZoneCenter.Y)
9C308
cTLMaxDisplay.Y=
cBRDisplayZone.Y+iMaxDisplayHeight;
ELSE
cTLMaxDisplay.Y= cTLDisplayZone.Y;
END_IF;
9C310
rSession.cTLMaxDisplay = cTLMaxDisplay;
RETURN

FIGURE 9C3

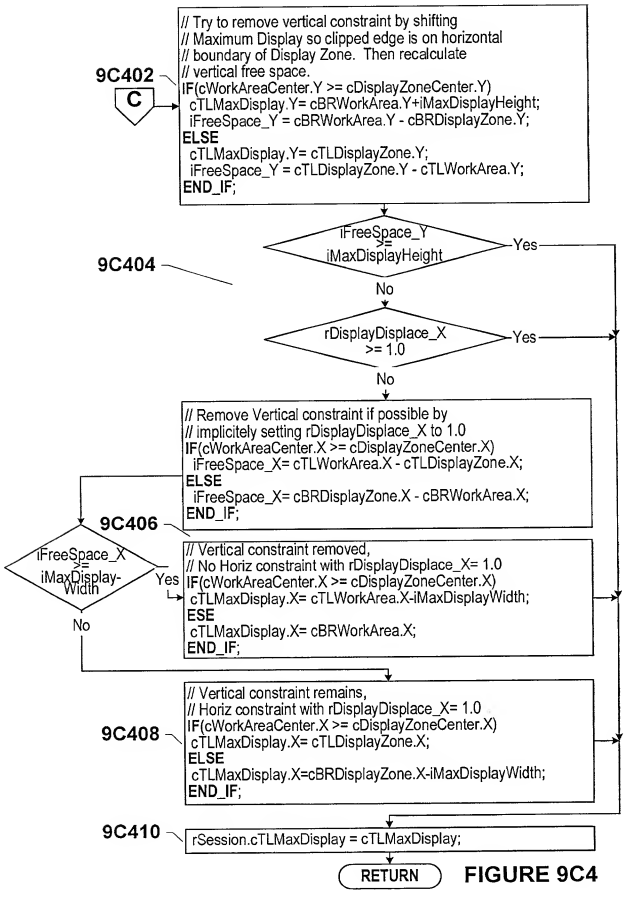

9C402
C
// Try to remove vertical constraint by shifting
// Maximum Display so clipped edge is on horizontal
// boundary of Display Zone. Then recalculate
// vertical free space.
IF(cWorkAreaCenter.Y >= cDisplayZoneCenter.Y)
cTLMaxDisplay.Y= cBRWorkArea.Y+iMaxDisplayHeight;
iFreeSpace_Y = cBRWorkArea.Y - cBRDisplayZone.Y;
ELSE
cTLMaxDisplay.Y= cTLDisplayZone.Y;
iFreeSpace_Y = cTLDisplayZone.Y - cTLWorkArea.Y;
END_IF;
9C404
iFreeSpace_Y >= iMaxDisplayHeight
Yes
No
rDisplayDisplace_X >= 1.0
Yes
No
// Remove Vertical constraint if possible by
// implicitely setting rDisplayDisplace_X to 1.0
IF(cWorkAreaCenter.X >= cDisplayZoneCenter.X)
iFreeSpace_X= cTLWorkArea.X - cTLDisplayZone.X;
ELSE
iFreeSpace_X= cBRDisplayZone.X - cBRWorkArea.X;
END_IF;
9C406
iFreeSpace_X >= iMaxDisplay-Width
Yes
No
// Vertical constraint removed,
// No Horiz constraint with rDisplayDisplace_X= 1.0
IF(cWorkAreaCenter.X >= cDisplayZoneCenter.X)
cTLMaxDisplay.X= cTLWorkArea.X-iMaxDisplayWidth;
ESE
cTLMaxDisplay.X= cBRWorkArea.X;
END_IF;
// Vertical constraint remains,
// Horiz constraint with rDisplayDisplace_X= 1.0
9C408
IF(cWorkAreaCenter.X >= cDisplayZoneCenter.X)
cTLMaxDisplay.X= cTLDisplayZone.X;
ELSE
cTLMaxDisplay.X=cBRDisplayZone.X-iMaxDisplayWidth;
END_IF;
9C410
rSession.cTLMaxDisplay = cTLMaxDisplay;
RETURN

FIGURE 9C4

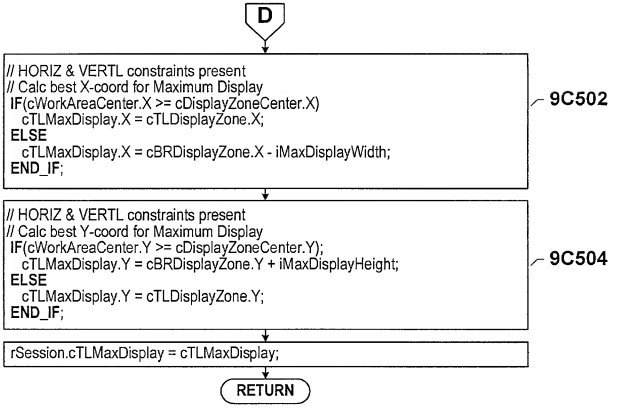


FIGURE 9C5


PROCEDURE CalculateFixedPointCoordinates(
sControlType, rDisplayParms, rSession, llnCD);
9D00
// Fetch needed values from data structures
rFixedPointDisplace_X =
rDisplayParms.aControl[sControlType].rVar.rFixedPointDisplace_X;
rFixedPointDisplace_Y =
rDisplayParms.aControl[sControlType].rVar.rFixedPointDisplace_Y;
iMaxDisplayHeight= rSession.iMaxDisplayHeight;
iMaxShellWidth = rSession.iMaxShellWidth;
iMaxShellHeight = rSession.iMaxShellHeight;
cTLMaxDisplay = rSession.cTLMaxDisplay;
9D02
// Determine top-left corner of the maximum shell
cTLMaxShell.X= cTLMaxDisplay.X;
cTLMaxShell.Y= cTLMaxDisplay.Y-(iMaxDisplayHeight-iMaxShellHeight);
9D04
// Determine Fixed-Point Coordinates
pnCD = rSession.pnCD;
*pnCD.rCurrentDisplay.cFixedPoint.X =
cTLMaxShell.X + (rFixedPointDisplace_X*iMaxShellWidth);
*pnCD.rCurrentDisplay.cFixedPoint.Y =
cTLMaxShell.Y - (rFixedPointDisplace_Y*iMaxShellHeight);
9D06
// Translate axis to position the axis origin at the fixed-point
TRANSLATE_AXIS(FROM = rSessionParms.cInitialAxisOrigin,
TO =*pnCD.rCurrentDisplay.cFixedPoint);
9D08
// Determine defining corners of display zone under translated axis
cTLDisplayZone = GET_DISPLAY_ZONE_ORIGIN();
iDisplayZoneWidth= GET_DISPLAY_ZONE_WIDTH();
iDisplayZoneHeight= GET_DISPLAY_ZONE_HEIGHT();
cBRDisplayZone.X= cTLDisplayZone.X + iDisplayZoneWidth;
cBRDisplayZone.Y= cTLDisplayZone.Y - iDisplayZoneHeight;
RETURN

FIGURE 9D

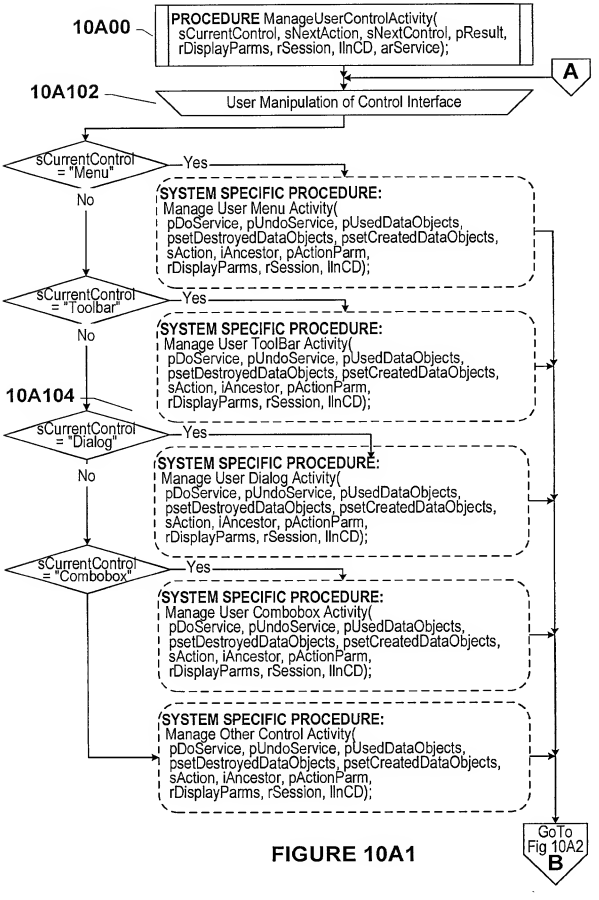


FIGURE 10A1

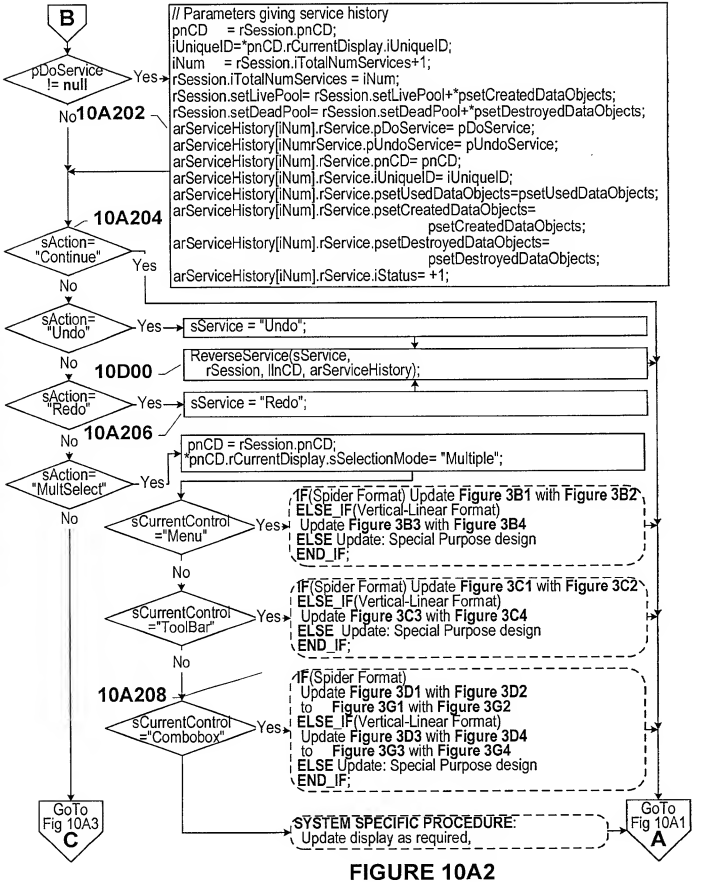


FIGURE 10A2

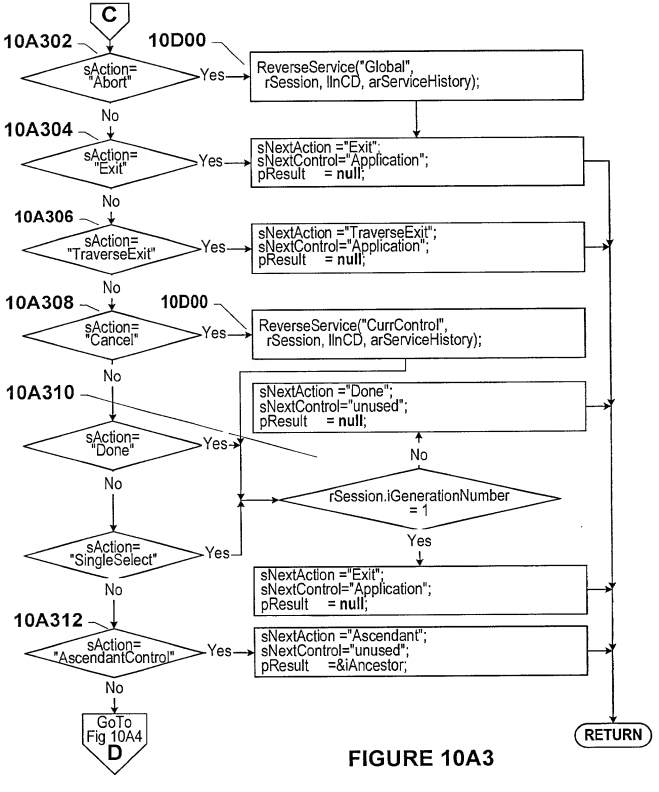

C
10A302
sAction= "Abort"
Yes
10D00
ReverseService("Global", rSession, lInCD, arServiceHistory);
No
10A304
sAction= "Exit"
Yes
sNextAction ="Exit";
sNextControl="Application";
pResult = null;
No
10A306
sAction= "TraverseExit"
Yes
sNextAction ="TraverseExit";
sNextControl="Application";
pResult = null;
No
10A308
sAction= "Cancel"
Yes
10D00
ReverseService("CurrControl", rSession, lInCD, arServiceHistory);
No
10A310
sAction= "Done"
Yes
sNextAction ="Done";
sNextControl="unused";
pResult = null;
No
rSession.iGenerationNumber = 1
Yes
No
sAction= "SingleSelect"
Yes
sNextAction ="Exit";
sNextControl="Application";
pResult = null;
No
10A312
sAction= "AscendantControl"
Yes
sNextAction ="Ascendant";
sNextControl="unused";
pResult =&iAncestor;
No
GoTo Fig 10A4
D
RETURN

FIGURE 10A3

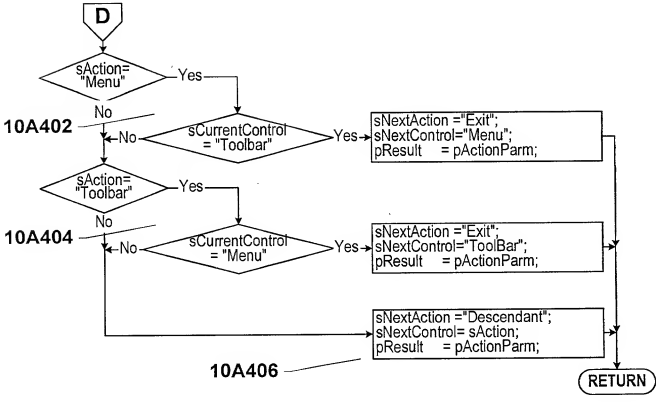


FIGURE 10A4

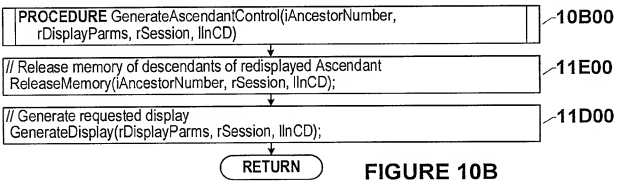


FIGURE 10B

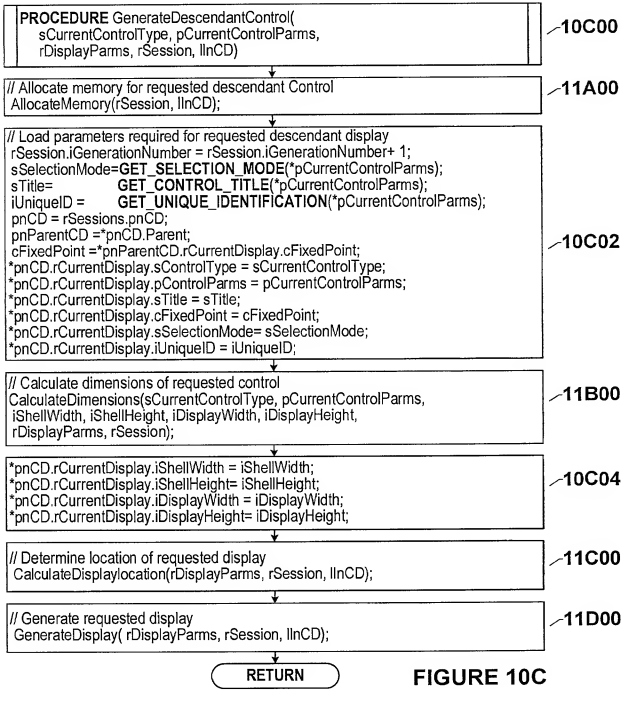


FIGURE 10C

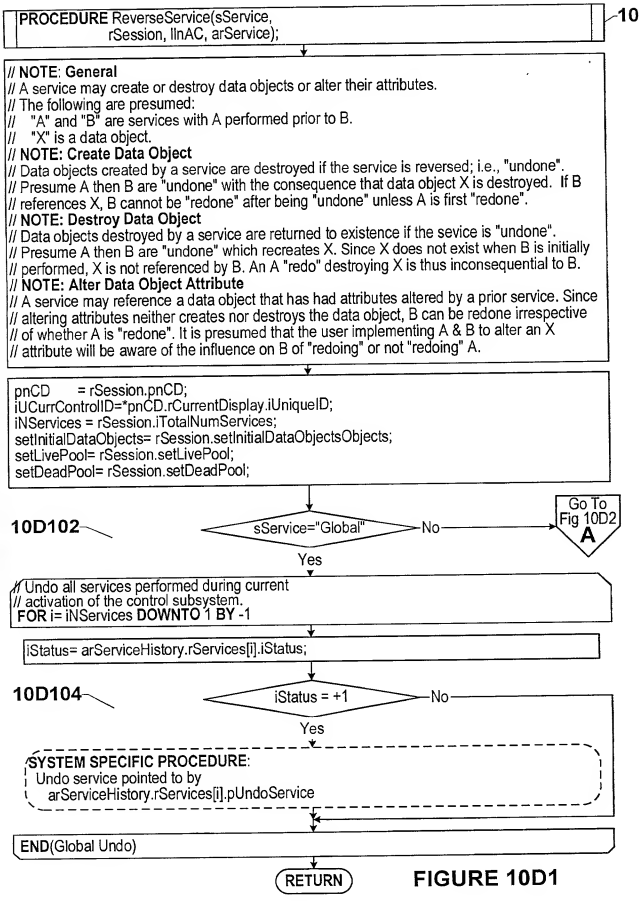

PROCEDURE ReverseService(sService, rSession, IInAC, arService);
10D00
// NOTE: General
// A service may create or destroy data objects or alter their attributes.
// The following are presumed:
// "A" and "B" are services with A performed prior to B.
// "X" is a data object.
// NOTE: Create Data Object
// Data objects created by a service are destroyed if the service is reversed; i.e., "undone".
// Presume A then B are "undone" with the consequence that data object X is destroyed. If B
// references X, B cannot be "redone" after being "undone" unless A is first "redone".
// NOTE: Destroy Data Object
// Data objects destroyed by a service are returned to existence if the sevice is "undone".
// Presume A then B are "undone" which recreates X. Since X does not exist when B is initially
// performed, X is not referenced by B. An A "redo" destroying X is thus inconsequential to B.
// NOTE: Alter Data Object Attribute
// A service may reference a data object that has had attributes altered by a prior service. Since
// altering attributes neither creates nor destroys the data object, B can be redone irrespective
// of whether A is "redone". It is presumed that the user implementing A & B to alter an X
// attribute will be aware of the influence on B of "redoing" or not "redoing" A.
pnCD = rSession.pnCD;
iUCurrControlID=*pnCD.rCurrentDisplay.iUniqueID;
iNServices = rSession.iTotalNumServices;
setInitialDataObjects= rSession.setInitialDataObjectsObjects;
setLivePool= rSession.setLivePool;
setDeadPool= rSession.setDeadPool;
10D102
sService="Global"
No
Go To Fig 10D2 A
Yes
// Undo all services performed during current
// activation of the control subsystem.
FOR i= iNServices DOWNTO 1 BY -1
iStatus= arServiceHistory.rServices[i].iStatus;
10D104
iStatus = +1
No
Yes
SYSTEM SPECIFIC PROCEDURE:
Undo service pointed to by
arServiceHistory.rServices[i].pUndoService
END(Global Undo)
RETURN

FIGURE 10D1

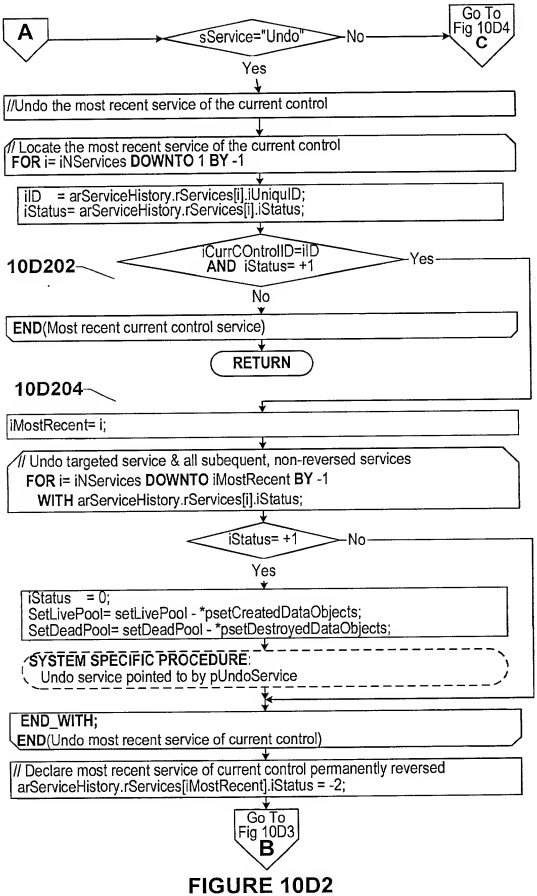

A
sService="Undo"
No
Go To Fig 10D4 C
Yes
//Undo the most recent service of the current control
// Locate the most recent service of the current control
FOR i= iNServices DOWNTO 1 BY -1
iID = arServiceHistory.rServices[i].iUniqueID;
iStatus= arServiceHistory.rServices[i].iStatus;
10D202
iCurrCOntrolID=iID AND iStatus= +1
Yes
No
END(Most recent current control service)
RETURN
10D204
iMostRecent= i;
// Undo targeted service & all subequent, non-reversed services
FOR i= iNServices DOWNTO iMostRecent BY -1
WITH arServiceHistory.rServices[i].iStatus;
iStatus= +1
No
Yes
iStatus = 0;
SetLivePool= setLivePool - *psetCreatedDataObjects;
SetDeadPool= setDeadPool - *psetDestroyedDataObjects;
SYSTEM SPECIFIC PROCEDURE:
Undo service pointed to by pUndoService
END_WITH;
END(Undo most recent service of current control)
// Declare most recent service of current control permanently reversed
arServiceHistory.rServices[iMostRecent].iStatus = -2;
Go To Fig 10D3 B

FIGURE 10D2

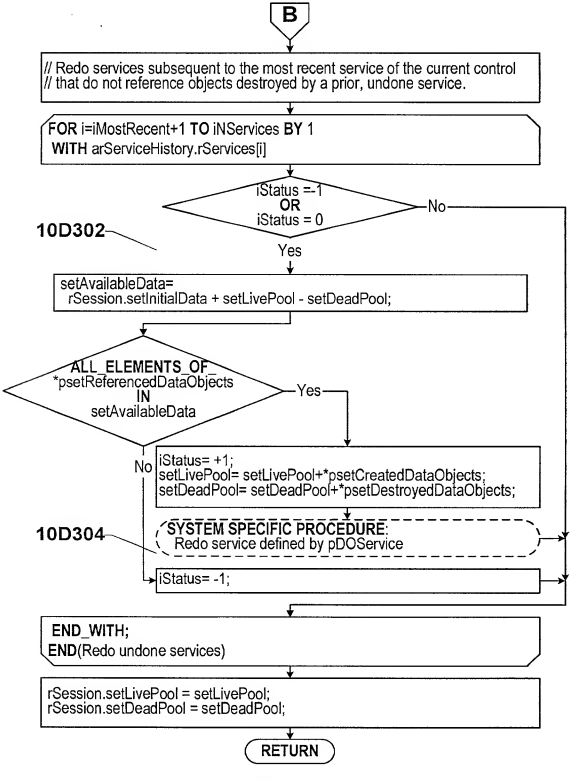

B
// Redo services subsequent to the most recent service of the current control
// that do not reference objects destroyed by a prior, undone service.
FOR i=iMostRecent+1 TO iNServices BY 1
WITH arServiceHistory.rServices[i]
iStatus =-1
OR
iStatus = 0
No
10D302
Yes
setAvailableData=
rSession.setInitialData + setLivePool - setDeadPool;
ALL_ELEMENTS_OF
*psetReferencedDataObjects
IN
setAvailableData
Yes
No
iStatus= +1;
setLivePool= setLivePool+*psetCreatedDataObjects;
setDeadPool= setDeadPool+*psetDestroyedDataObjects;
10D304
SYSTEM SPECIFIC PROCEDURE:
Redo service defined by pDOService
iStatus= -1;
END_WITH;
END(Redo undone services)
rSession.setLivePool = setLivePool;
rSession.setDeadPool = setDeadPool;
RETURN

FIGURE 10D3

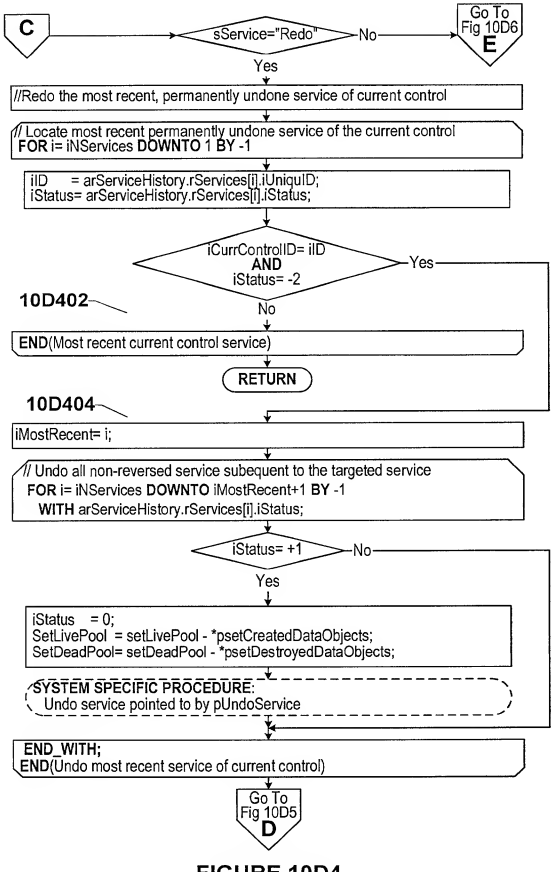

C
sService="Redo"
No
Go To Fig 10D6 E
Yes
//Redo the most recent, permanently undone service of current control
// Locate most recent permanently undone service of the current control
FOR i= iNServices DOWNTO 1 BY -1
iID = arServiceHistory.rServices[i].iUniqueID;
iStatus= arServiceHistory.rServices[i].iStatus;
iCurrControlID= iID AND iStatus= -2
Yes
10D402
No
END(Most recent current control service)
RETURN
10D404
iMostRecent= i;
// Undo all non-reversed service subequent to the targeted service
FOR i= iNServices DOWNTO iMostRecent+1 BY -1
WITH arServiceHistory.rServices[i].iStatus;
iStatus= +1
No
Yes
iStatus = 0;
SetLivePool = setLivePool - *psetCreatedDataObjects;
SetDeadPool= setDeadPool - *psetDestroyedDataObjects;
SYSTEM SPECIFIC PROCEDURE:
Undo service pointed to by pUndoService
END_WITH;
END(Undo most recent service of current control)
Go To Fig 10D5 D

FIGURE 10D4

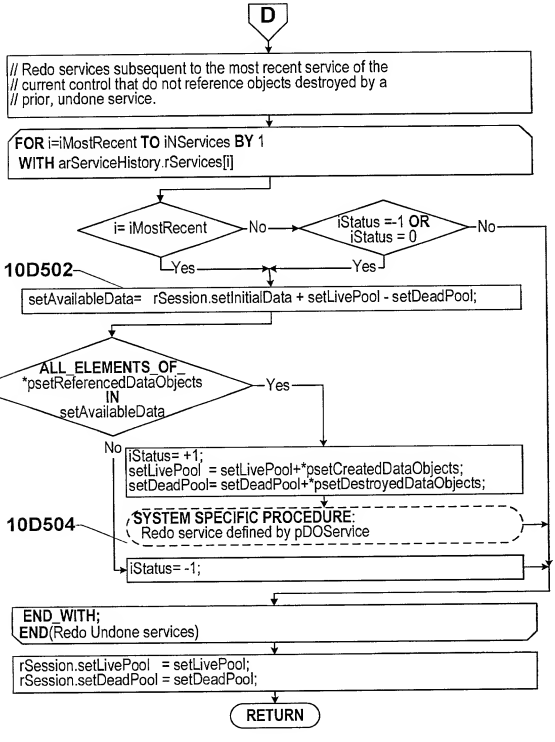

D
// Redo services subsequent to the most recent service of the
// current control that do not reference objects destroyed by a
// prior, undone service.
FOR i=iMostRecent TO iNServices BY 1
WITH arServiceHistory.rServices[i]
i= iMostRecent
No
iStatus =-1 OR
iStatus = 0
No
Yes
Yes
10D502
setAvailableData= rSession.setInitialData + setLivePool - setDeadPool;
ALL_ELEMENTS_OF_
*psetReferencedDataObjects
IN
setAvailableData
Yes
No
iStatus= +1;
setLivePool = setLivePool+*psetCreatedDataObjects;
setDeadPool= setDeadPool+*psetDestroyedDataObjects;
10D504
SYSTEM SPECIFIC PROCEDURE:
Redo service defined by pDOService
iStatus= -1;
END_WITH;
END(Redo Undone services)
rSession.setLivePool = setLivePool;
rSession.setDeadPool = setDeadPool;
RETURN

FIGURE 10D5

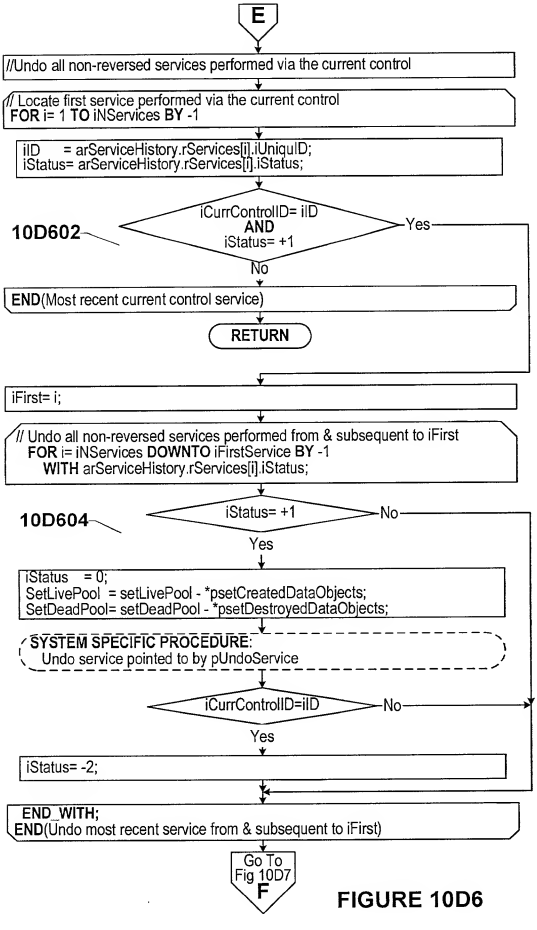

E
//Undo all non-reversed services performed via the current control
// Locate first service performed via the current control
FOR i= 1 TO iNServices BY -1
iID = arServiceHistory.rServices[i].iUniquID;
iStatus= arServiceHistory.rServices[i].iStatus;
10D602
iCurrControlID= iID
AND
iStatus= +1
Yes
No
END(Most recent current control service)
RETURN
iFirst= i;
// Undo all non-reversed services performed from & subsequent to iFirst
FOR i= iNServices DOWNTO iFirstService BY -1
WITH arServiceHistory.rServices[i] iStatus;
10D604
iStatus= +1
No
Yes
iStatus = 0;
SetLivePool = setLivePool - *psetCreatedDataObjects;
SetDeadPool= setDeadPool - *psetDestroyedDataObjects;
SYSTEM SPECIFIC PROCEDURE:
Undo service pointed to by pUndoService
iCurrControlID=iID
No
Yes
iStatus= -2;
END_WITH;
END(Undo most recent service from & subsequent to iFirst)
Go To
Fig 10D7
F

FIGURE 10D6

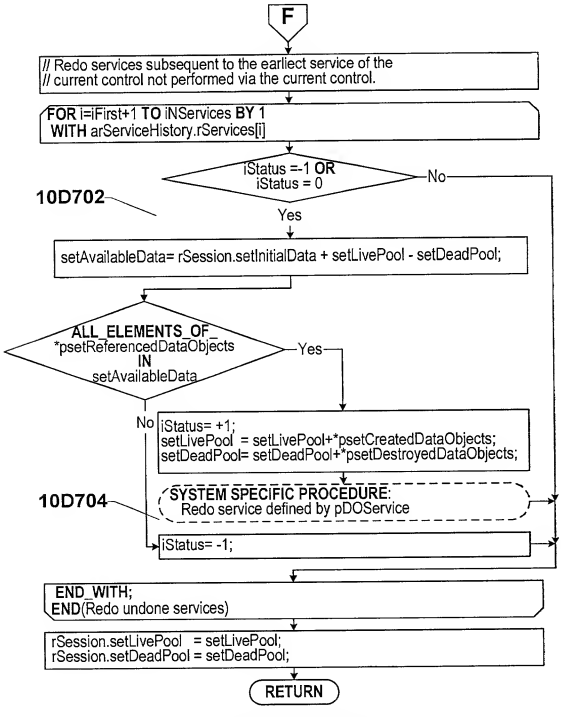

F
// Redo services subsequent to the earliest service of the
// current control not performed via the current control.
FOR I=iFirst+1 TO iNServices BY 1
WITH arServiceHistory.rServices[i]
iStatus =-1 OR
iStatus = 0
No
10D702
Yes
setAvailableData= rSession.setInitialData + setLivePool - setDeadPool;
ALL_ELEMENTS_OF_
*psetReferencedDataObjects
IN
setAvailableData
Yes
No
iStatus= +1;
setLivePool = setLivePool+*psetCreatedDataObjects;
setDeadPool= setDeadPool+*psetDestroyedDataObjects;
10D704
SYSTEM SPECIFIC PROCEDURE:
Redo service defined by pDOService
iStatus= -1;
END_WITH;
END(Redo undone services)
rSession.setLivePool = setLivePool;
rSession.setDeadPool = setDeadPool;
RETURN

FIGURE 10D7

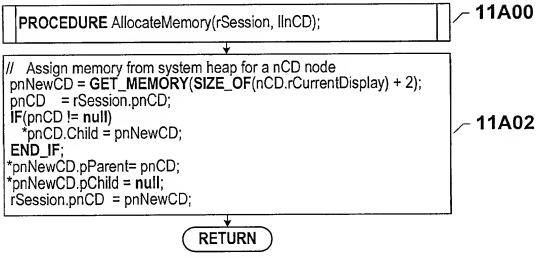

PROCEDURE AllocateMemory(rSession, llnCD);
11A00
// Assign memory from system heap for a nCD node
pnNewCD = GET_MEMORY(SIZE_OF(nCD.rCurrentDisplay) + 2);
pnCD = rSession.pnCD;
IF(pnCD != null)
*pnCD.Child = pnNewCD;
END_IF;
*pnNewCD.pParent= pnCD;
*pnNewCD.pChild = null;
rSession.pnCD = pnNewCD;
11A02
RETURN

FIGURE 11A

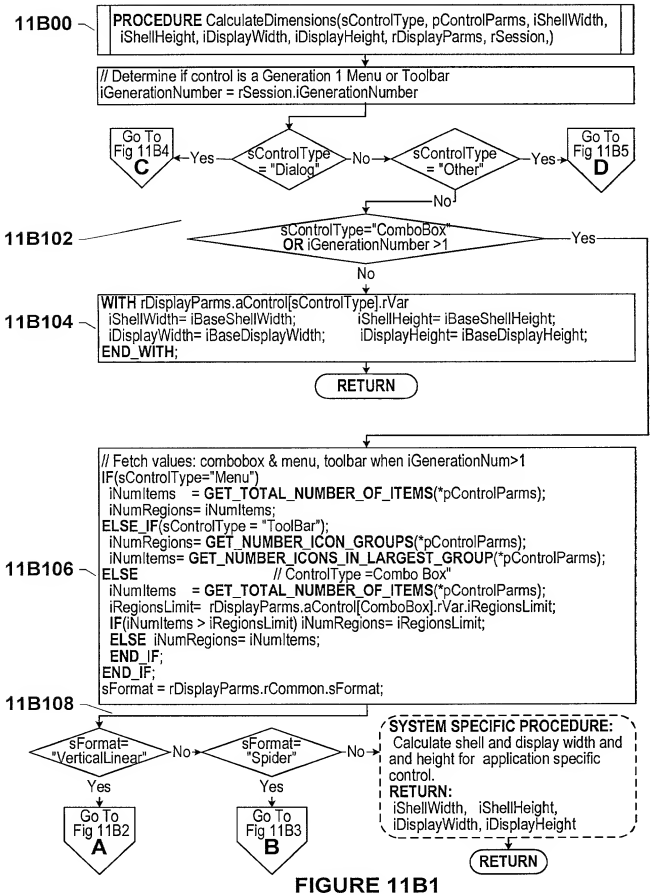

11B00
PROCEDURE CalculateDimensions(sControlType, pControlParms, iShellWidth, iShellHeight, iDisplayWidth, iDisplayHeight, rDisplayParms, rSession,)
// Determine if control is a Generation 1 Menu or Toolbar
iGenerationNumber = rSession.iGenerationNumber
sControlType = "Dialog"
Yes
Go To Fig 11B4 C
No
sControlType = "Other"
Yes
Go To Fig 11B5 D
No
11B102
sControlType="ComboBox" OR iGenerationNumber >1
Yes
No
11B104
WITH rDisplayParms.aControl[sControlType].rVar
iShellWidth= iBaseShellWidth; iShellHeight= iBaseShellHeight;
iDisplayWidth= iBaseDisplayWidth; iDisplayHeight= iBaseDisplayHeight;
END_WITH;
RETURN
11B106
// Fetch values: combobox & menu, toolbar when iGenerationNum>1
IF(sControlType="Menu")
iNumItems = GET_TOTAL_NUMBER_OF_ITEMS(*pControlParms);
iNumRegions= iNumItems;
ELSE_IF(sControlType = "ToolBar");
iNumRegions= GET_NUMBER_ICON_GROUPS(*pControlParms);
iNumItems= GET_NUMBER_ICONS_IN_LARGEST_GROUP(*pControlParms);
ELSE // ControlType =Combo Box"
iNumItems = GET_TOTAL_NUMBER_OF_ITEMS(*pControlParms);
iRegionsLimit= rDisplayParms.aControl[ComboBox].rVar.iRegionsLimit;
IF(iNumItems > iRegionsLimit) iNumRegions= iRegionsLimit;
ELSE iNumRegions= iNumItems;
END_IF;
END_IF;
sFormat = rDisplayParms.rCommon.sFormat;
11B108
sFormat= "VerticalLinear"
No
sFormat= "Spider"
No
Yes
Go To Fig 11B2 A
Yes
Go To Fig 11B3 B
SYSTEM SPECIFIC PROCEDURE:
Calculate shell and display width and and height for application specific control.
RETURN:
iShellWidth, iShellHeight,
iDisplayWidth, iDisplayHeight
RETURN

FIGURE 11B1

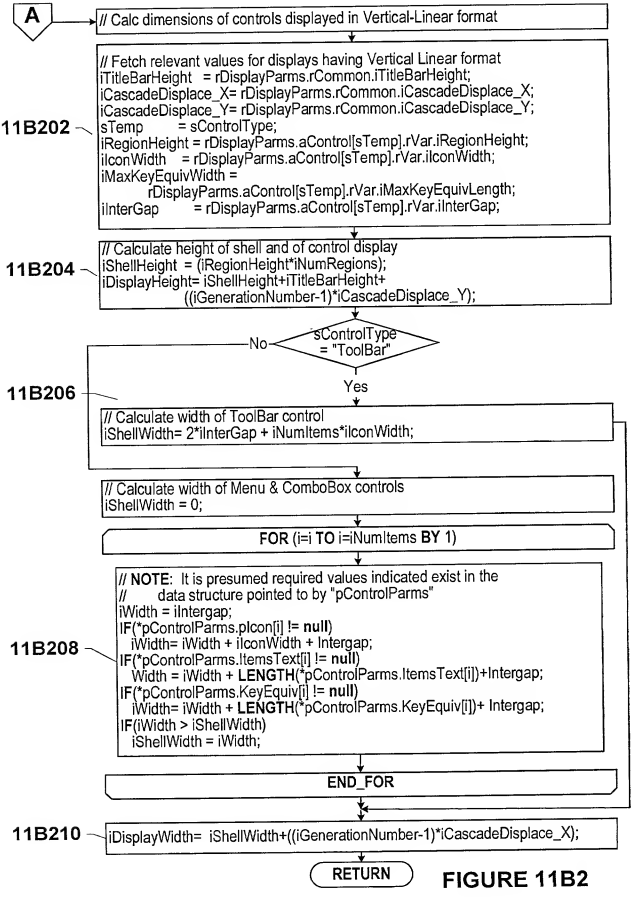

A
// Calc dimensions of controls displayed in Vertical-Linear format
// Fetch relevant values for displays having Vertical Linear format
iTitleBarHeight = rDisplayParms.rCommon.iTitleBarHeight;
iCascadeDisplace_X= rDisplayParms.rCommon.iCascadeDisplace_X;
iCascadeDisplace_Y= rDisplayParms.rCommon.iCascadeDisplace_Y;
11B202
sTemp = sControlType;
iRegionHeight = rDisplayParms.aControl[sTemp].rVar.iRegionHeight;
iIconWidth = rDisplayParms.aControl[sTemp].rVar.iIconWidth;
iMaxKeyEquivWidth =
rDisplayParms.aControl[sTemp].rVar.iMaxKeyEquivLength;
iInterGap = rDisplayParms.aControl[sTemp].rVar.iInterGap;
// Calculate height of shell and of control display
11B204
iShellHeight = (iRegionHeight*iNumRegions);
iDisplayHeight= iShellHeight+iTitleBarHeight+
((iGenerationNumber-1)*iCascadeDisplace_Y);
No
sControlType = "ToolBar"
Yes
11B206
// Calculate width of ToolBar control
iShellWidth= 2*iInterGap + iNumItems*iIconWidth;
// Calculate width of Menu & ComboBox controls
iShellWidth = 0;
FOR (i=i TO i=iNumItems BY 1)
// NOTE: It is presumed required values indicated exist in the
// data structure pointed to by "pControlParms"
iWidth = iIntergap;
11B208
IF(*pControlParms.pIcon[i] != null)
iWidth= iWidth + iIconWidth + Intergap;
IF(*pControlParms.ItemsText[i] != null)
Width = iWidth + LENGTH(*pControlParms.ItemsText[i])+Intergap;
IF(*pControlParms.KeyEquiv[i] != null)
iWidth= iWidth + LENGTH(*pControlParms.KeyEquiv[i])+ Intergap;
IF(iWidth > iShellWidth)
iShellWidth = iWidth;
END_FOR
11B210
iDisplayWidth= iShellWidth+((iGenerationNumber-1)*iCascadeDisplace_X);
RETURN

FIGURE 11B2

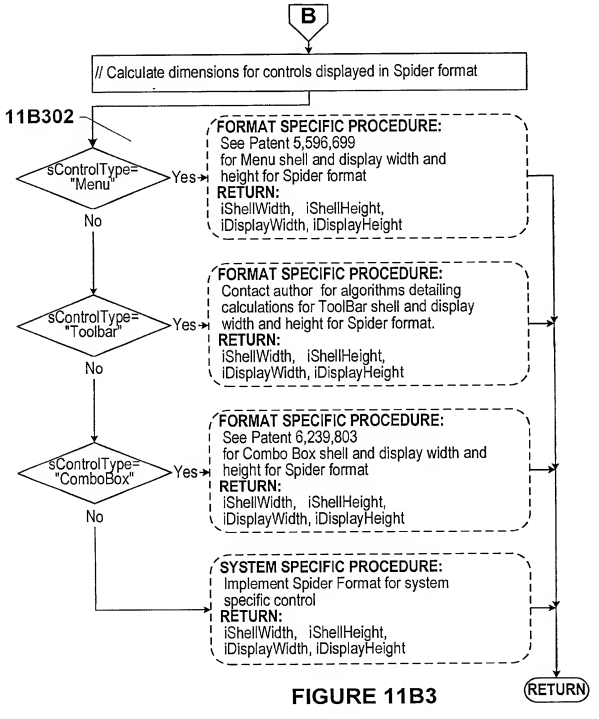


FIGURE 11B3

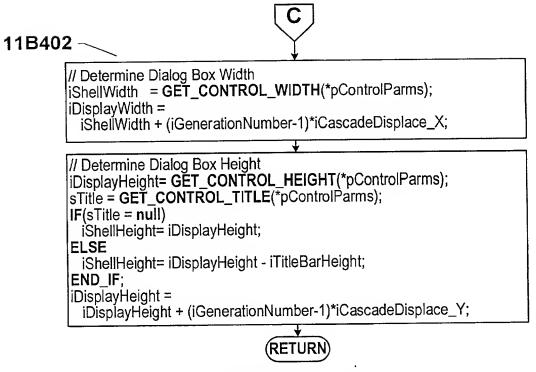


FIGURE 11B4

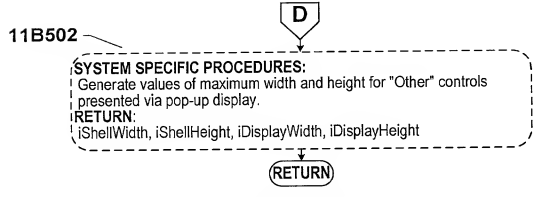


FIGURE 11B5

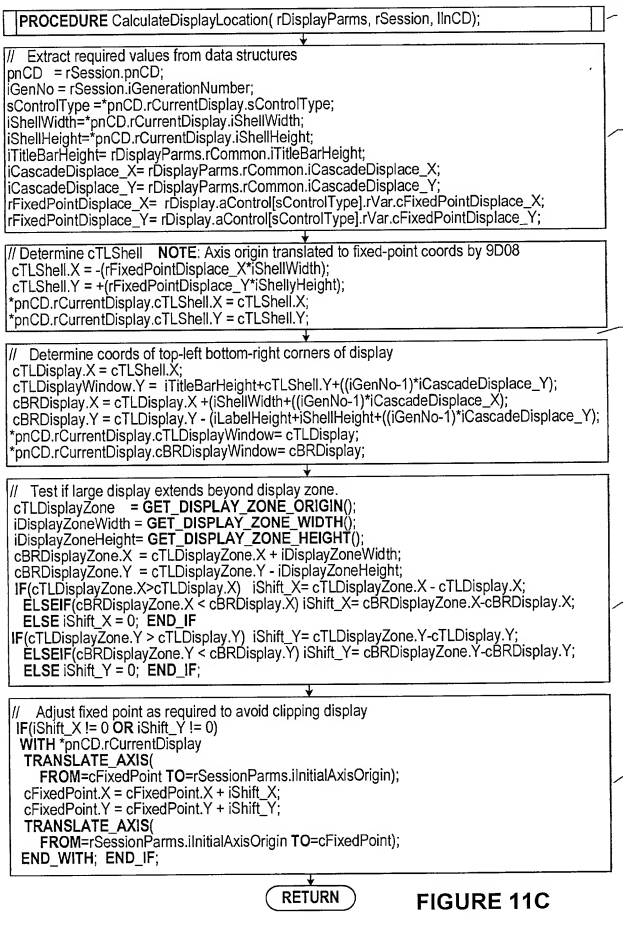

PROCEDURE CalculateDisplayLocation(rDisplayParms, rSession, llnCD);
11C00
// Extract required values from data structures
pnCD = rSession.pnCD;
iGenNo = rSession.iGenerationNumber;
sControlType =*pnCD.rCurrentDisplay.sControlType;
iShellWidth=*pnCD.rCurrentDisplay.iShellWidth;
iShellHeight=*pnCD.rCurrentDisplay.iShellHeight;
iTitleBarHeight= rDisplayParms.rCommon.iTitleBarHeight;
iCascadeDisplace_X= rDisplayParms.rCommon.iCascadeDisplace_X;
iCascadeDisplace_Y= rDisplayParms.rCommon.iCascadeDisplace_Y;
rFixedPointDisplace_X= rDisplay.aControl[sControlType].rVar.cFixedPointDisplace_X;
rFixedPointDisplace_Y= rDisplay.aControl[sControlType].rVar.cFixedPointDisplace_Y;
11C02
// Determine cTLShell NOTE: Axis origin translated to fixed-point coords by 9D08
cTLShell.X = -(rFixedPointDisplace_X*iShellWidth);
cTLShell.Y = +(rFixedPointDisplace_Y*iShellyHeight);
*pnCD.rCurrentDisplay.cTLShell.X = cTLShell.X;
*pnCD.rCurrentDisplay.cTLShell.Y = cTLShell.Y;
11C04
// Determine coords of top-left bottom-right corners of display
cTLDisplay.X = cTLShell.X;
cTLDisplayWindow.Y = iTitleBarHeight+cTLShell.Y+((iGenNo-1)*iCascadeDisplace_Y);
cBRDisplay.X = cTLDisplay.X +(iShellWidth+((iGenNo-1)*iCascadeDisplace_X);
cBRDisplay.Y = cTLDisplay.Y - (iLabelHeight+iShellHeight+((iGenNo-1)*iCascadeDisplace_Y);
*pnCD.rCurrentDisplay.cTLDisplayWindow= cTLDisplay;
*pnCD.rCurrentDisplay.cBRDisplayWindow= cBRDisplay;
// Test if large display extends beyond display zone.
cTLDisplayZone = GET_DISPLAY_ZONE_ORIGIN();
iDisplayZoneWidth = GET_DISPLAY_ZONE_WIDTH();
iDisplayZoneHeight= GET_DISPLAY_ZONE_HEIGHT();
cBRDisplayZone.X = cTLDisplayZone.X + iDisplayZoneWidth;
cBRDisplayZone.Y = cTLDisplayZone.Y - iDisplayZoneHeight;
IF(cTLDisplayZone.X>cTLDisplay.X) iShift_X= cTLDisplayZone.X - cTLDisplay.X;
ELSEIF(cBRDisplayZone.X < cBRDisplay.X) iShift_X= cBRDisplayZone.X-cBRDisplay.X;
ELSE iShift_X = 0; END_IF
IF(cTLDisplayZone.Y > cTLDisplay.Y) iShift_Y= cTLDisplayZone.Y-cTLDisplay.Y;
ELSEIF(cBRDisplayZone.Y < cBRDisplay.Y) iShift_Y= cBRDisplayZone.Y-cBRDisplay.Y;
ELSE iShift_Y = 0; END_IF;
11C06
// Adjust fixed point as required to avoid clipping display
IF(iShift_X != 0 OR iShift_Y != 0)
WITH *pnCD.rCurrentDisplay
TRANSLATE_AXIS(
FROM=cFixedPoint TO=rSessionParms.iInitialAxisOrigin);
cFixedPoint.X = cFixedPoint.X + iShift_X;
cFixedPoint.Y = cFixedPoint.Y + iShift_Y;
TRANSLATE_AXIS(
FROM=rSessionParms.iInitialAxisOrigin TO=cFixedPoint);
END_WITH; END_IF;
11C08
RETURN

FIGURE 11C

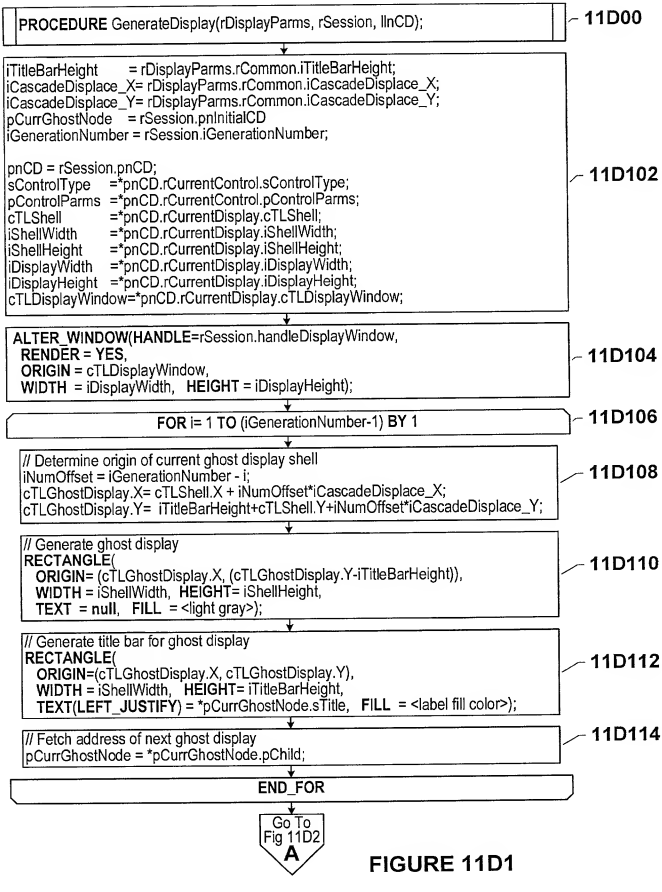

PROCEDURE GenerateDisplay(rDisplayParms, rSession, InCD);
11D00
iTitleBarHeight = rDisplayParms.rCommon.iTitleBarHeight;
iCascadeDisplace_X= rDisplayParms.rCommon.iCascadeDisplace_X;
iCascadeDisplace_Y= rDisplayParms.rCommon.iCascadeDisplace_Y;
pCurrGhostNode = rSession.pnInitialCD
iGenerationNumber = rSession.iGenerationNumber;
pnCD = rSession.pnCD;
sControlType =*pnCD.rCurrentControl.sControlType;
pControlParms =*pnCD.rCurrentControl.pControlParms;
cTLShell =*pnCD.rCurrentDisplay.cTLShell;
iShellWidth =*pnCD.rCurrentDisplay.iShellWidth;
iShellHeight =*pnCD.rCurrentDisplay.iShellHeight;
iDisplayWidth =*pnCD.rCurrentDisplay.iDisplayWidth;
iDisplayHeight =*pnCD.rCurrentDisplay.iDisplayHeight;
cTLDisplayWindow=*pnCD.rCurrentDisplay.cTLDisplayWindow;
11D102
ALTER_WINDOW(HANDLE=rSession.handleDisplayWindow,
RENDER = YES,
ORIGIN = cTLDisplayWindow,
WIDTH = iDisplayWidth, HEIGHT = iDisplayHeight);
11D104
FOR i= 1 TO (iGenerationNumber-1) BY 1
11D106
// Determine origin of current ghost display shell
iNumOffset = iGenerationNumber - i;
cTLGhostDisplay.X= cTLShell.X + iNumOffset*iCascadeDisplace_X;
cTLGhostDisplay.Y= iTitleBarHeight+cTLShell.Y+iNumOffset*iCascadeDisplace_Y;
11D108
// Generate ghost display
RECTANGLE(
ORIGIN= (cTLGhostDisplay.X, (cTLGhostDisplay.Y-iTitleBarHeight)),
WIDTH = iShellWidth, HEIGHT= iShellHeight,
TEXT = null, FILL = <light gray>);
11D110
// Generate title bar for ghost display
RECTANGLE(
ORIGIN=(cTLGhostDisplay.X, cTLGhostDisplay.Y),
WIDTH = iShellWidth, HEIGHT= iTitleBarHeight,
TEXT(LEFT_JUSTIFY) = *pCurrGhostNode.sTitle, FILL = <label fill color>);
11D112
// Fetch address of next ghost display
pCurrGhostNode = *pCurrGhostNode.pChild;
11D114
END_FOR
Go To Fig 11D2 A

FIGURE 11D1

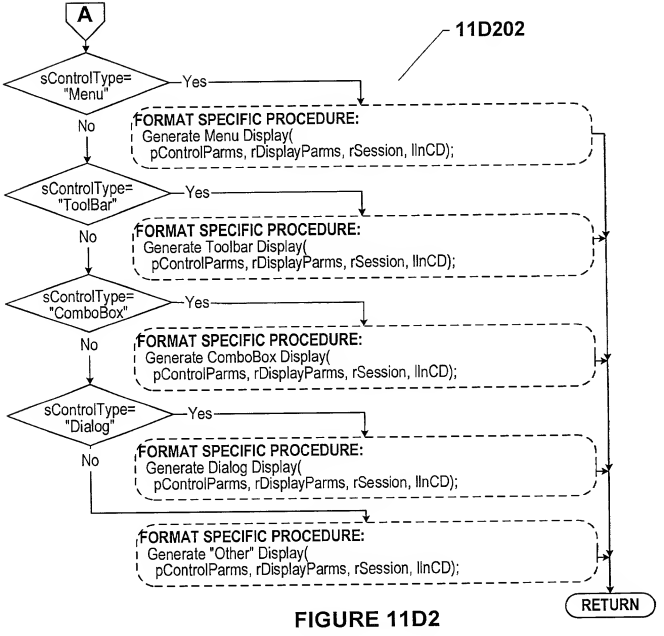

A
11D202
sControlType= "Menu"
Yes
No
FORMAT SPECIFIC PROCEDURE:
Generate Menu Display(
pControlParms, rDisplayParms, rSession, llnCD);
sControlType= "ToolBar"
Yes
No
FORMAT SPECIFIC PROCEDURE:
Generate Toolbar Display(
pControlParms, rDisplayParms, rSession, llnCD);
sControlType= "ComboBox"
Yes
No
FORMAT SPECIFIC PROCEDURE:
Generate ComboBox Display(
pControlParms, rDisplayParms, rSession, llnCD);
sControlType= "Dialog"
Yes
No
FORMAT SPECIFIC PROCEDURE:
Generate Dialog Display(
pControlParms, rDisplayParms, rSession, llnCD);
FORMAT SPECIFIC PROCEDURE:
Generate "Other" Display(
pControlParms, rDisplayParms, rSession, llnCD);
RETURN

FIGURE 11D2

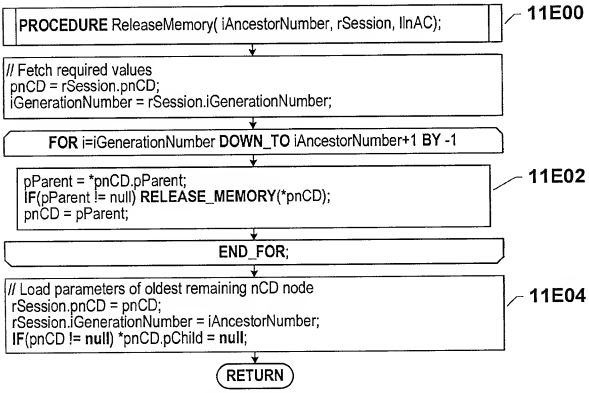


FIGURE 11E

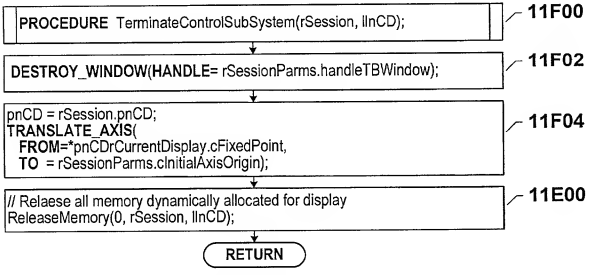


FIGURE 11F